Karlsruhe Institute of Technology

Algorithmen zur Intuitiven Manipulation Geometrischer Graphen

Layout-Adaptierung für Drag&Drop-Operationen

Studienarbeit
von

Christian Wellenbrock

An der Fakultät für Informatik
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Gutachter:	Prof. Dr. Dorothea Wagner
Betreuender Mitarbeiter:	Ignaz Rutter
Zweiter betreuender Mitarbeiter:	Marcus Krug

Bearbeitungszeit: 16. August 2011 – 15. Februar 2012

Zusammenfassung

Bei der Visualisierung großer Graphen stoßen automatische Layoutalgorithmen schnell an ihre Grenzen. Will man etwa bestimmte Strukturen, Subgraphen oder Symmetrien hervorheben, müssen oft große Teile des Graphen manipuliert werden. Mit heutigen Editoren ist dies oft nur
Um bestimmte Strukturen, Subgraphen und Symmetrien herauszuarbeiten, sind große Transformationen mit heutigen Editoren oft nur durch Verschieben einzelner oder Gruppen von Knoten zu bewerkstelligen. In dieser Arbeit soll es darum gehen, den Benutzer bei solchen Operationen interaktiv zu unterstützen. Wir werden uns konkret mit dem Problem befassen, in einem Graphen Platz für einen vorgegebenes Polygon zu schaffen, um dort etwa einen weiteren Teilgraphen einzufügen.
Zunächst untersuchen wir die Komplexität dreier Varianten dieses Problems: Auf planaren Graphen, auf nicht notwendigerweise planaren Graphen und eine monotone Variante, die die Erhaltung der orthogonalen Kantenordnungen fordert. Wie sich herausstellen wird, sind all diese Probleme $\mathcal{NP}$-schwer.
Anschließend stellen wir zwei heuristische Verfahren vor. Die nichtuniforme Skalierung basiert auf einer interpolierten Skalierung der Ebene, die sehr schnell sehr brauchbare Ergebnisse liefert. Der Spring-Embedder ist ein kräftebasiertes Verfahren und liefert etwas bessere Ergebnisse bei deutlich längeren Laufzeiten.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Kalsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Datum	Unterschrift

Inhaltsverzeichnis

1. Einleitung

Graphen gehören zu den vielseitigsten Modellen zur Beschreibung von Strukturen, Zusammenhängen und Abläufen. Allein in der Informatik spielen sie in sehr vielen Disziplinen eine wichtige Rollen. Sie beschreiben Datenstrukturen, Kontrollflüsse, Kommunikations- und Navigationsnetze. Aber auch in der Physik, Chemie, Biologie, Soziologie und Linguistik basieren viele Verfahren auf graphentheoretischen Erkenntnissen.

Da man die internen Strukturen von konkreten Graphen und Graphklassen aus der Anwendung erfassen und für Verfahrensoptimierung verwenden möchte, spielen übersichtliche Visualisierungen verschiedenster Graphen eine sehr wichtige Rolle. Dabei versucht man die subjektive Güte einer Zeichnung durch Kennzahlen wie die Anzahl der Kantenkreuzungen, Knotenverteilung und Kantenlängenverteilung zu quantifizieren [LLY06, DH96]. Die meisten Verfahren suchen dabei Knotenpositionen und zeichnen Kanten als Strecken zwischen inzidenten Knoten. Für allgemeine Graphen lassen sich mit kräftebasierten und *simulated annealing* Verfahren recht gute Layouts ermitteln. Für konkretere Graphklassen wie gerichtete azyklische oder planare Graphen gibt es spezielle Verfahren, die etwa die Planarität ausnutzen und Kantenkreuzungen vermeiden können.

Klassische Layout-Algorithmen liefern gute Ergebnisse für statische Graphen. Kleine Änderungen am Graphen können aber zu sehr unterschiedlichen Zeichnungen führen. Die Diskrepanzen reichen von einfachen Drehungen bis hin zu komplett unterschiedlichen kombinatorischen Einbettungen, die vom Betrachter nur schwer nachvollzogen werden können, da die *mental map* nicht aufrecht erhalten wird. Bei Änderungen wie einer einfachen Drehung kann die mental map durch animiertes Morphen erhalten werden. Bei kompletten Restrukturierungen hilft aber auch dieser Ansatz nicht.

Bei der mental map handelt es sich dabei um Modelle zur Bewertung der Wiedererkennbarkeit des Graphen in angepassten Layouts. Nachdem man sich in einer Zeichnung zurechtgefunden hat, entwickelt man ein Gefühl dafür, welche Teile des Graphen sich wo in der Zeichnung befinden. Um diese so gelernten Annahmen auch nach einer Graphänderung möglichst aufrecht zu erhalten, versucht man etwa die horizontalen und vertikalen Knotenordnungen, die relativen Knotenabstände und die Graphtopologie beizubehalten [ELMS95, LLY06, FR06, DGK01, DG02].

Dynamische Layout-Algorithmen sind zusätzlich zur Optimierung der bereits angesprochenen Gütekriterien auch bemüht, diese mental map zu erhalten [BW97, CDBT+92]. Nach einer Änderung des Graphen lässt sich so eine Folgezeichnung generieren, die der vorherigen sehr ähnlich ist. Mit einem Morphing kann diese Änderung nun intuitiv nachvollziehbar dargestellt werden [FE02, EKP04].

In der Praxis sind diese Layout-Algorithmen unverzichtbar. Die meisten Graphen sind so groß, dass manuelle Layouts viel zu aufwändig wären. Oft kennt man auch die Struktur der Graphen gar nicht und möchte sie durch automatische Visualisierungen erst erkennen. Trotzdem werden auch die besten Algorithmen wohl nie die Zeichnungen liefern können, die der Autor im Sinn hat. Von einem automatisch generierten Layout ausgehend sind oft mehr oder weniger drastische Korrekturen notwendig. Abhängig von der Domäne und dem einzelnen Graphen können Symmetrien, bekannte Subgraphen oder andere besondere Strukturen herausgearbeitet werden wollen. Dieser Prozess der Nachbearbeitung ist mit den heute gängigen Werkzeugen oft mit erheblichem Aufwand verbunden, da diese meist nur die Translation einzelner oder mehrerer Knoten erlauben. Will man also etwa einen interessanten Teilgraphen vergrößern oder verschieben, müssen zunächst störende Knoten aus dem Zielbereich hinausgeschoben werden. Dann kann die eigentliche Transformation vollzogen werden. Anschließend muss gegebenenfalls zusätzlich im Randbereich aufgeräumt werden, um neu entstandene Kantenkreuzungen und Knotenhäufungen aufzulösen. Dabei kann es auch von Interesse sein, unabhängige Strukturen dabei nicht zu zerstören, sondern die mental map zu erhalten.

Wir interessieren uns für Algorithmen die solche und ähnliche Layoutanpassungen intuitiv unterstützen, indem sie dem Anwender etwa das vorherige Platzschaffen und anschließende Aufräumen abnehmen. Die Interaktion besteht dann lediglich aus der eigentlichen Transformation eines Teilgraphen, die sich dann automatisch adaptiv in den Restgraphen fortsetzt. In dieser Arbeit betrachten wir exemplarisch das bereits erwähnte Problem des Platzschaffens: Wie vergrößert man am Besten eine bestimmte Facette eines Graphen, um darin Platz für ein gegebenes Polygon zu schaffen?

Im folgenden Abschnitt werden wir diese Problemstellung formalisieren. In Abschnitt 3 werden wir zeigen, dass dieses Problem und zwei naheliegende Varianten davon $\mathcal{NP}$-schwer sind. In Abschnitt 4 beschreiben und evaluieren wir dann abschließend zwei heuristische Algorithmen.

2. Expandieren von Facetten

Bevor wir das in der Einleitung motivierte Problem des Platzschaffens formalisieren können, wollen wir zunächst einige grundsätzliche Begriffe und Notationen definieren. Da wir ein geometrisches Graph-Problem betrachten, haben wir es mit Punkten, Strecken, Polygonen, Graphen und Einbettungen zu tun. Auch wenn nicht alle betrachteten Graphen planar sind, werden wir oft von Facetten sprechen. Im nichtplanaren Fall sind dabei die Flächen gemeint, in die die Ebene zerfällt, wenn man die Kanten und Knoten des eingebetteten Graphen entfernt. Dies sind also die Facetten des planarisierten Graphen, der entsteht wenn man jede Kantenkreuzung durch einen zusätzlichen Knoten ersetzt.

Ist p ein Polygon, so wollen wir das Innere des Polygons mit $\mathcal{I}(p) \subset \mathbb{R}^2$ bezeichnen. Für einen Punkt $u \in \mathbb{R}^2$ seit $T_u(p)$ das um u verschobene Polygon. Sind $G = (V, E)$ ein Graph mit Einbettung $\Phi : V \to \mathbb{R}^2$ und $e = (u, v) \in E$ eine Kante, so sei $\Phi(e) = \Phi(u)\Phi(v)$ die geradlinige Verbindungsstrecke von $\Phi(u)$ nach $\Phi(v)$. Es bezeichne $\text{len}_\Phi(e) = \text{len}(\Phi(e)) = \|\Phi(u) - \Phi(v)\|_2$ die euklidische Länge der eingebetteten Kante. Ist f eine Facette von G, so sei analog $\Phi(f)$ das Polygon auf das die Facette f durch Φ eingebettet wird.

Im Zentrum dieser Arbeit soll die planare Variante unseres Problems stehen:

Planar Extend Face (PEF)

Gegeben sei ein planarer Graph $G = (V, E)$ mit planarer Einbettung $\Phi : V \to \mathbb{R}^2$, eine Facette f und ein Polygon p. Gesucht ist eine planare Einbettung $\Psi : V \to \mathbb{R}^2$ mit den folgenden Eigenschaften:

1. Φ und Ψ sind kombinatorisch äquivalent.
2. p "passt" in $\Psi(f)$, es existiert also ein $u \in \mathbb{R}^2$ mit $\mathcal{I}(T_u(p)) \subset \mathcal{I}(\Psi(f))$.
3. Ψ minimiert die Gesamtkantenlängenänderung $\Delta(\Phi, \Psi)$ bezüglich Φ:

$$\Delta(\Phi, \Psi) = \sum_{e \in E} |\text{len}_\Phi(e) - \text{len}_\Psi(e)|$$

Da sich durch eine Streckung $\Psi = \lambda \cdot \Phi$ um einen genügend großen Faktor $\lambda \in \mathbb{R}$ leicht die Punkte 1 und 2 erfüllen lassen, ist dies ein wohldefiniertes Minimierungsproblem. Im folgenden Abschnitt werden wir dieses Problem, die nichtplanare und eine weitere monotone Variante auf ihre Komplexität untersuchen.

3. Komplexität

In diesem Kapitel zeigen wir die $\mathcal{NP}$-Schwere von Planar Extend Face und zwei weiteren Varianten.

3.1 Planares Expandieren

Satz 1 *PEF ist $\mathcal{NP}$-schwer.*

Beweis: Wir betrachten das zugehörige Entscheidungsproblem PEF′, ob es eine Einbettung Ψ gibt, die die Punkte 1, 2 und 3′ erfüllt:

3′. $\Delta(\Phi, \Psi) = 0$.

Wir beweisen die $\mathcal{NP}$-Schwere, indem wir vom $\mathcal{NP}$-vollständigen Problem PlanarMonotone3SAT reduzieren [KR92]. Eine Instanz $\varphi = (X, C)$ von PlanarMonotone3SAT ist eine 3SAT-Formel mit Variablen X und Klauseln $C = C^+ \cup C^-$. Dabei kommen in den Klauseln C^+ nur positive und in C^- nur negative Literale vor. Betrachtet man jede Variable und jede Klausel von φ als Knoten eines Graphen in dem jeder Klausel-Knoten genau mit den zugehörigen Variablen-Knoten verbunden ist, so erhält man den Variablen-Klausel-Graphen G_φ von φ, der für jede PlanarMonotone3SAT-Instanz planar ist. Dieser Variablen-Klausel-Graph lässt sich sogar immer so anordnen, dass alle Variablen-Knoten X auf der x-Achse, alle positiven Klausel-Knoten C^+ in der oberen Halbebene und alle negativen Klausel-Knoten C^- in der unteren Halbebene liegen [KR92]. Ein Beispiel ist in Abbildung 3.1 links zu sehen.

Wir konstruieren nun eine Instanz $\psi = (G, \Phi, f, p)$ von PEF′, die genau dann lösbar ist, wenn die PlanarMonotone3SAT-Formel φ erfüllbar ist. Das Ergebnis dieser Konstruktion für eine Beispiel-Instanz ist in Abbildung 3.1 rechts dargestellt.

Dazu legen wir zunächst für jeden Variablen-Knoten alle ausgehenden vertikalen Kantenstücke in einem gemeinsamen Kanal übereinander (Abbildung 3.1 in der Mitte) und konstruieren ψ indem wir Variablen, Klauseln und Verbindungskanten von

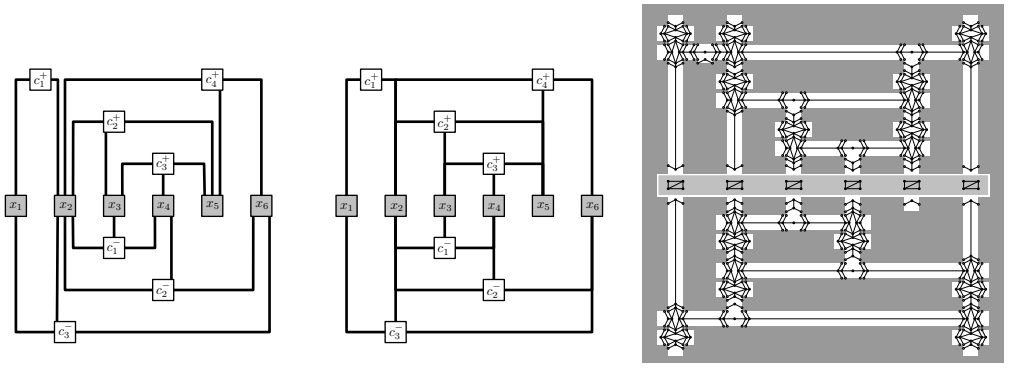

Abbildung 3.1: Der Variable-Klausel-Graph einer Instanz von PlanarMonotone3SAT (links), der gleiche Graph mit zusammengelegten Kanten (Mitte) und die von uns daraus konstruierte PEF′-Instanz (rechts).

φ durch geeignete Gadgets ersetzen. Die in Abbildung 3.1 dargestellten grauen Bereiche seien dabei in ihrem Inneren vollständig planar trianguliert. Da sich die Kantenlängen wegen Bedingung 3′ nicht ändern dürfen, ist jedes innere Dreieck dieser Triangulierung starr. Wegen Bedingung 1 ist auch die relative Lage dieser Dreiecke bestimmt und folglich ist jeder graue Bereich insgesamt starr bezüglich jeder zulässigen Einbettung Ψ. Wie sich herausstellen wird, ist auch die relative Anordnung benachbarter Bereiche eindeutig festgelegt. Somit können die grauen Bereiche und die grauen Knoten auf deren Rändern als global fixiert angenommen werden. Die übrigen beweglichen Knoten werden schwarz und das freizumachende Polygon p wird hellgrau dargestellt.

Wir beginnen die Konstruktion mit einem genügend breiten rechteckigen Polygon p in einer etwas größeren Facette f_p von G entlang der x-Achse, auf dem wir nebeneinander für jede Variable $x \in X$ ein Variablen-Gadget anordnen. Dieses in Abbildung 3.2 dargestellte Gadget besteht aus einem rechteckigen Teilgraphen der Höhe h und Breite b der im Inneren von p liegt und zwei Kanälen, die von f_p ausgehend nach oben und unten zeigen. Beide Kanäle haben ebenfalls Breite b und in einem Abstand von h ober- beziehungsweise unterhalb des Kanaleingangs befinden sich links und rechts je ein Ankerknoten. Die beiden linken Ankerknoten sind dabei jeweils durch einen Pfad aus zwei Kanten mit den entsprechenden rechten Ankerknoten verbunden, der insgesamt etwas länger ist als b. Der jeweils in der Mitte dieser Pfade liegende Brückenknoten ist dabei in Richtung des Polygons p geklappt. Die fett dargestellten vertikalen Kanten der Kanaleingänge und des Rechtecks sind dabei nur symbolische Stellvertreter für die in den Detailansichten dargestellten y-monotonen Kantenzüge. Diese in den umpunkteten und umstrichelten Bereichen dargestellten Ränder sind dabei so aufgebaut, dass sie genau dann zusammenpassen, wenn sie zum gleichen Variablen-Gadget gehören. Ein Beispiel für solch ein passendes Paar ist links und rechts in Abbildung 3.2 dargestellt. Die schwarz dargestellten Knoten oben und unten dienen dabei der relativen Ausrichtung beider Muster. Die weißen Knoten in der Mitte kodieren dabei je ein Bit vom Index i der zugehörigen Variable $x_i \in X$ durch seine x-Koordinate. Ist etwa das least significant bit gesetzt (i also ungerade), so wird der oberste weiße Knoten rechts angeordnet. Um alle Variablen eindeutig zu kodieren sind dabei $\log |X|$ weiße Knoten notwendig. Die grauen Bereiche und das innere Rechteck seinen mit diesen speziellen Rändern geeignet trianguliert um

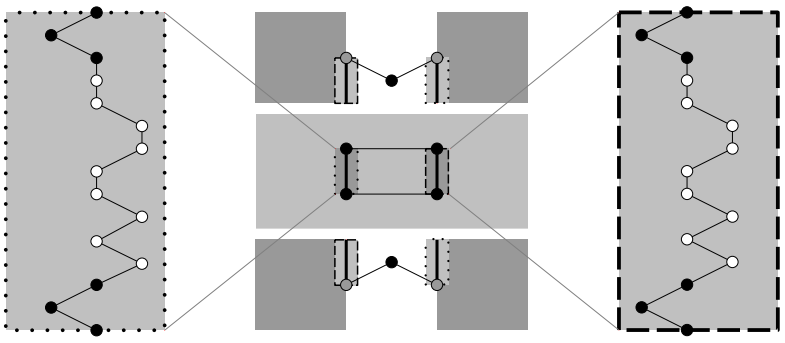

Abbildung 3.2: Ein Variablen-Gadget (Mitte) und Detailansichten der umpunkteten (links) und umstrichelten Bereiche (rechts). Die weißen Knoten kodieren dabei den Variablenindex von 514.

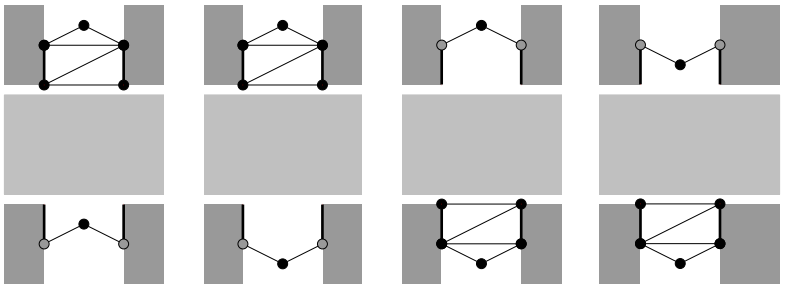

Abbildung 3.3: Die vier zulässigen Konfiguration eines Variablen-Gadgets.

Verformungen zu verhindern.

Da die grauen Knoten fixiert sind und wir alle Kantenlängen erhalten wollen, können die beiden damit verbundenen schwarzen Knoten höchstens vom freizumachenden Polygon "wegklappen". Dabei wird jeweils ein Bereich frei, der nach Konstruktion gerade groß genug ist um das zugehörige Rechteck aufzunehmen. Da das Rechteck nur in die beiden zum gleichen Variablen-Gadget gehörenden Slots passt und wir das hellgraue Polygon freimachen wollen, muss also wenigstens einer der beiden schwarzen Brückenknoten nach außen klappen um Platz zu machen. Die vier möglichen Konfigurationen sind in Abbildung 3.3 dargestellt.

Die nach oben oder unten geklappten Brückenknoten interpretieren wir im Folgenden als Signale, die von den Variablen-Gadgets ausgehen. Wir sprechen dann von Signalen, die von der Variable weg, oder zur Variablen hin zeigen. Für zulässige Konfigurationen halten wir fest:

A1: Von den beiden von einem Variablen-Gadget ausgehenden Signalen zeigt mindestens eins von der Variable weg.

Mit den in Abbildung 3.4 dargestellten Kanten-Gadgets lassen sich diese Signale beliebig weit entlang der Achsenrichtungen propagieren. Sie bestehen aus einem beliebig langen horizontalen oder vertikalen Kanal mit Durchmesser b und zwei durch eine Kante verbundene Brückenknoten wie wir sie schon im Variablen-Gadget gesehen haben. Wir ersetzen jedes geradlinige Kantenstück des Variable-Klausel-Graphen G_φ

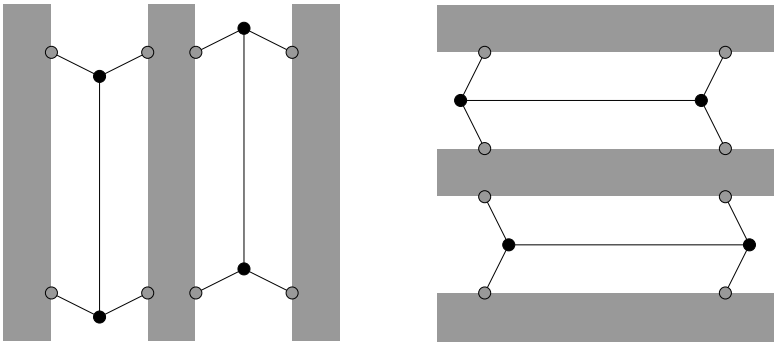

Abbildung 3.4: Die vertikalen (links) und horizontalen (rechts) Kanten-Gadgets in ihren zulässigen Konfigurationen.

durch solch einen passenden Kanal. Da die grauen Knoten fixiert sind, gibt es für jeden schwarzen Knoten wieder genau zwei mögliche Positionen. Durch die Kante entlang des Kanals sind jeweils beide Brückenknoten gekoppelt und es ergeben sich die dargestellten Konfigurationen.

Das in Abbildung 3.5 dargestellte Kreuz-Gadget dient dazu, ein von oben oder unten kommendes Signal nach links und rechts weiterzugeben. Diese Kreuzung von einem horizontalen und einem vertikalen Kanal besitzt ein Kanten-Gadget an jeder der vier Kanalmündungen, von denen zwei gegenüberliegende zur Kreuzung hin- und die beiden anderen von der Kreuzung weggeklappt sind. Die vier jeweils innersten Brückenknoten definieren somit eine Raute die wir mit zusätzlichen vier Kanten fixieren. Jede dieser Kante bildet mit zwei inneren Brückenkanten und einer Kanalecke somit ein rechtwinkliges gleichschenkliges Dreieck. Damit legt ein aus einem Kanal kommendes Signal eindeutig die in die drei anderen Kanäle ausgehenden Signale fest.

Wir nehmen nun für einen Moment an, dass lediglich die beiden oberen grauen Bereiche fixiert sind. Damit ist das obere Kanten-Gadget entweder hoch- oder runtergeklappt. Durch die beiden oberen Dreiecks-Facetten werden die inneren Brückenknoten der linken und rechten Kanten-Gadgets nach innen oder außen gedrückt. Die Parallelogramm-Facette in der Mitte des Gadgets legt nun die Position des inneren Brückenknotens des unteren Kanten-Gadgets und die anliegenden unteren Dreiecke legen die Eckpunkte der unteren grauen Bereiche fest. Aus Symmetriegründen sind diese in beiden Fällen gleich plaziert. Durch die vier Parallelogramm-Facetten Kanten-Gadgets links und rechts ist auch die Ausrichtung der unteren grauen Bereiche gleich der der oberen und damit ist auch das untere Kanten-Gadget festgelegt. Zusammenfassend gilt also:

A2: Sind zwei benachbarte graue Bereiche des Kreuz-Gadgets fixiert, so sind es auch die beiden anderen.

Wie man der Abbildung 3.5 entnehmen kann, werden einkommende Signale im folgenden Sinne weitergeleitet: Zeigt das von oben eingehende Signal von der Variablen weg, so auch die ausgehenden Signale links und rechts. Analoges gilt für von unten kommende und zur Variablen hin zeigende Signale. Da das auf der gegenüberliegenden Seite ausgehende Signal gerade invertiert ist, können wir zwei Kreuz-Gadgets zu

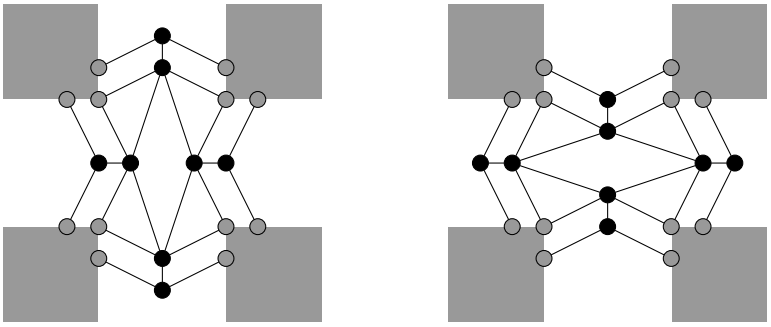

Abbildung 3.5: Das Kreuz-Gadget in seinen zwei möglichen Konfigurationen.

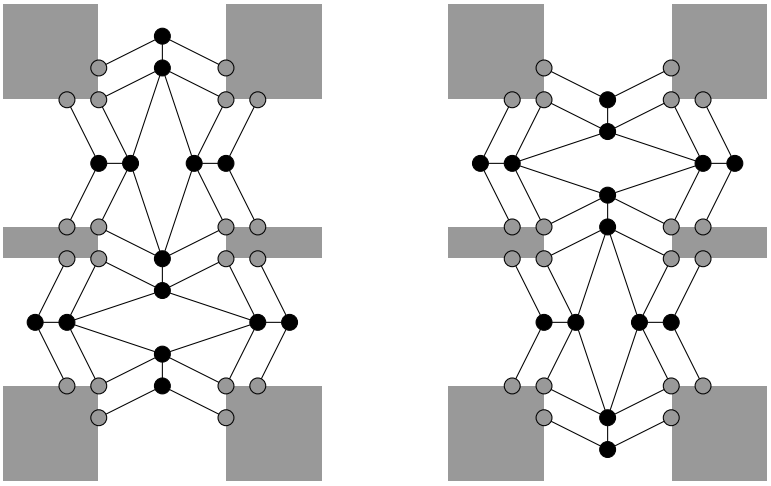

Abbildung 3.6: Das Verteiler-Gadget in beiden Konfigurationen.

einem in Abbildung 3.6 dargestellten Verteiler-Gadget kombinieren, indem wir das untere Kanten-Gadget des einen mit dem oberen Kanten-Gadget des anderen identifizieren. Damit wird das oben oder unten angelegte Signal nun korrekt in alle drei anderen Richtungen weitergibt. Nach Konstruktion sind dabei die seitlichen Ausgänge zu verwenden, die näher am Variablen-Gadget liegen. Dieses Verteiler-Gadget ist an allen Kreuzungen und Knicken der Kanten des Variable-Klausel-Graphen G_φ anzubringen um die geradlinigen Kanten-Gadgets zu verbinden. Aus der Aussage A2 folgt für alle zulässigen Einbettungen:

A3: Sind zwei benachbarte graue Bereiche des Kreuz-Gadgets fixiert, so sind es auch die vier anderen.

Um die Konstruktion abzuschließen, brauchen wir noch das in Abbildung 3.7 dargestellte Klausel-Gadget, das alle Klauseln im Variable-Klausel-Graphen G_φ ersetzt. Für Klauseln in C^+ kommen die Signale von unten, links und rechts. Für Klauseln aus C^- kommen die Signale entsprechend von oben, links und rechts. Da die Signale des Variablen-Gadgets in Grundstellung zur Variablen hin zeigen und die Kanten- und Verteiler-Gadgets diese entsprechend weiterleiten, zeigen auch die am Klausel-Gadget ankommenden Signale in der Grundstellung zu den Variablen hin, also vom

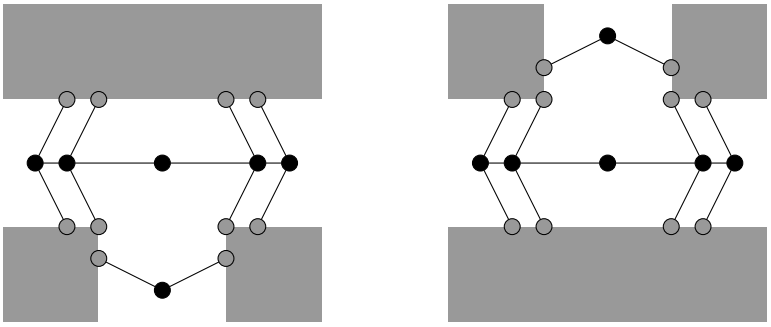

Abbildung 3.7: Das Klausel-Gadget für Klauseln aus C^+ (links) und Klauseln aus C^- (rechts).

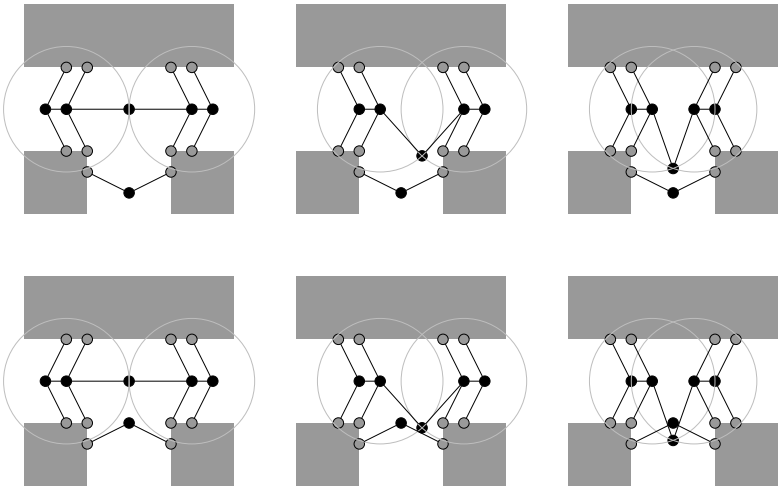

Abbildung 3.8: Die sechs konzeptionell verschiedenen Konfigurationen des Klausel-Gadgets (wobei die letzte nicht zulässig ist).

Klausel-Gadget weg.

Von den möglichen acht Konfigurationen des C^+-Gadgets sind sechs in Abbildung 3.8 dargestellt. Die übrigen zwei sind symmetrisch zu denen in der Mitte und auch die Konfigurationen des C^--Gadgets sind lediglich gespiegelt. Wir stellen fest, dass nur die Konfiguration für drei von den Variablen weg zeigende Signale nicht planar eingebettet werden kann (unten rechts) und folgern:

A4: Ein Klausel-Gadget kann genau dann planar eingebettet werden, wenn wenigstens ein eingehendes Signal zur Variable hin zeigt.

Für Zweierklauseln kann der mittlere Eingang des Klausel-Gadgets wie in Abbildung 3.9 nach Innen gezwungen werden und die Aussage A4 gilt auch für diese speziellen Klausel-Gadgets.

Unsere Konstruktion besteht also darin, den Variable-Klausel-Graphen G_φ durch die vorgestellten Gadgets zu ersetzen. Bevor wir uns der Korrektheit der Reduktion widmen, beweisen wir zunächst die eingangs gemachte Annahme, dass alle grauen Bereiche des Graphen G auch relativ zueinander starr sind.

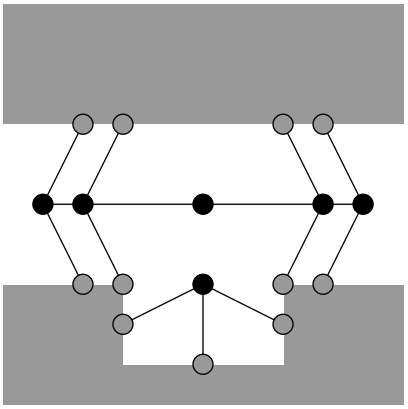

Abbildung 3.9: Das Klausel-Gadget mit eingeschränktem mittlerem Eingang.

Wir betrachten die durch die Kanten des Variable-Klausel-Graphen G_φ und die x-Achse induzierte Partitionierung der Ebene. Da sich benachbarte Partitionen immer in mindestens einer Kantenkreuzung oder -ecke berühren, sind benachbarte graue Bereiche im konstruierten Graphen G immer über mindestens ein Verteiler-Gadget verbunden. Ausgehend von dem äußeren alles umschließenden grauen Bereich folgt die relative Starrheit der benachbarten Bereiche nun mit Aussage A3, da von jedem anliegende Verteiler-Gadget mindestens zwei "seiner" grauen Bereiche im äußeren grauen Bereich liegen. Da diese Argumentation auch für die Nachbarn dieser Nachbarn gilt, folgt induktiv die Starrheit aller grauen Bereiche.

Für die Korrektheit dieser Reduktion zeigen wir nun, dass die konstruierte PEF′-Instanz ψ genau dann lösbar ist, wenn die PlanarMonotone3SAT-Formel φ erfüllbar ist. Für die erste Implikation betrachten wir eine Einbettung Ψ, die ψ löst. Gemäß obiger Überlegungen ist jedes Gadget in einem der jeweils dargestellten Konfigurationen. Für die Variablen-Gadgets gilt wie in Abbildung 3.3 dargestellt, dass sich die inneren Rechtecke jeweils entweder im oberen oder unteren Slot befinden. Dies induziert im folgenden Sinne eine Variablenbelegung: Jede Variable x_i ist genau dann wahr, wenn sich das innere Rechteck des zugehörigen Variablen-Gadgets im unteren Slot befindet (Abbildung 3.3 rechts). Wir zeigen nun, dass diese Belegung die PlanarMonotone3SAT-Formel φ erfüllt.

Sei zunächst $c^+ \in C^+$ eine beliebige Klausel mit positiven Literalen. Nach Konstruktion ist das zugehörige Klausel-Gadget mit den Oberseiten der zugehörigen Variablen-Gadgets verbunden. Da die Einbettung Ψ zulässig ist, zeigt von diesem Klausel-Gadget nach Aussage A4 mindestens ein Signal zur Variable hin. Über Kanten- und Verteiler-Gadgets weitergeleitet, ist also der obere Brückenknoten des entsprechenden Variablen-Gadgets nach unten geklappt, und folglich befindet sich dessen inneres Rechteck im unteren Slot. Nach Wahl der Variablenbelegung ist die entsprechende Variable wahr, und somit ist auch die Klausel c^+ erfüllt.

Für eine beliebige Klausel $c^- \in C^-$ mit negierten Literalen gilt analog: Das zugehörige Klausel-Gadget ist mit den Unterseiten der zugehörigen Variablen-Gadgets verbunden und wenigstens ein Signal zeigt zur Variablen hin. Somit ist der untere Brückenknoten des Variablen-Gadgets nach oben geklappt und das Rechteck kann sich nur im oberen Slot befinden. Also ist diese Variable falsch, das negative Lite-

ral wahr und schließlich die Klausel c^- erfüllt. Zusammenfassend ist die vorgestellte Variablenbelegung also erfüllend für die Instanz φ.

Für die umgekehrte Implikation betrachten wir eine Variablenbelegung, die die PlanarMonotone3SAT-Formel φ erfüllt. Für jede wahre (falsche) Variable wählen wir für das zugehörige Variablen-Gadget die in Abbildung 3.3 rechts (links) dargestellte Konfiguration in der der obere (untere) Brückenknoten nach unten (oben) geklappt ist. Da die entsprechenden Signale über die Kanten- und Verteiler-Gadgets zu den Klausel-Gadgets weitergeleitet werden, ist damit eine komplette Einbettung Ψ definiert. Es bleibt zu zeigen, dass kein Klausel-Gadget die nicht planare Konfiguration annimmt.

Sei wieder $c^+ \in C^+$ eine beliebige Klausel mit positiven Literalen. Da die entsprechende Klausel nach Wahl der Variablenbelegung erfüllt ist, ist mindestens eine der zugehörigen Variablen wahr. Nach Wahl der Variablen-Gadget-Konfiguration zeigt das nach oben ausgehende Signal zur Variablen hin. Da dieses Signal zum betrachteten Klausel-Gadget geleitet wird, zeigt mindestens eines der eingehenden Signale zur Variablen hin. Nach Aussage A4 ist diese Konfiguration zulässig.

Für eine beliebige Klausel $c^- \in C^-$ mit negierten Literalen gilt analog: Da die Klausel erfüllt ist, ist wenigstens ein Literal positiv, also mindestens eine zugehörige Variable negativ. Nach Konstruktion ist der untere Knoten des entsprechenden Variablen-Gadgets nach oben geklappt und dieses am Klausel-Gadget eingehende Signal zeigt wieder zu Variable hin. Mit Aussage A4 folgt wieder die Zulässigkeit dieser Konfiguration. Insgesamt löst damit die konstruierte Einbettung Ψ die Instanz ψ.

Insgesamt ist damit die Korrektheit der beschriebenen Reduktion gezeigt. Da wir den Variable-Klausel-Graph G_φ in Polynomialzeit berechnen können [dBK08] und anschließend nur jedes seiner linear vielen Bausteine durch unsere beschränkt großen Gadgets ersetzen, ist diese Konstruktion auch in Polynomialzeit durchführbar und es folgt die $\mathcal{NP}$-Schwere von PEF. ■

Bemerkung: Das Problem PEF bleibt auch dann noch $\mathcal{NP}$-schwer, wenn man auf die kombinatorische Äquivalenz der Einbettungen Φ und Ψ verzichtet. Der Beweis kann mit den folgenden Anpassungen analog geführt werden. Zum Einen muss bei der Triangulierung der grauen Bereiche darauf geachtet werden, dass diese Einbettung die einzige planare ist. Dies kann durch Einfügen innerer Steiner-Knoten erreicht werden. Des Weiteren muss verhindert werden, dass die inneren Rechtecke der Variablen-Gadgets nicht die umschließende Facette f verlassen können. Dies kann beispielsweise dadurch erreicht werden, dass jedes Rechteck r mit einem genügend langen "Seil" an einem gemeinsamen Ankerknoten a von f befestigt wird. Dieser Pfad sollte lang und feingliedrig genug sein, um das Rechteck r in beiden möglichen Positionen mit den Knoten a verbinden zu können und dabei innerhalb von f, aber außerhalb des freizumachenden Rechtecks p zu verlaufen. Dieser Sachverhalt ist für eine Beispielkonfiguration in Abbildung 3.10 dargestellt.

3.2 Allgemeines Expandieren

Verzichtet man auf die Planarität des Graphen G und der Einbettungen Φ und Ψ, so ergibt sich analog das folgende Problem:

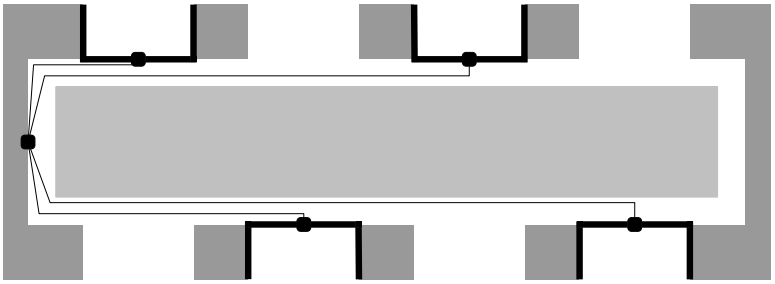

Abbildung 3.10: Die Rechtecke der Variablen-Gadgets sind mit dem Ankerknoten links verbunden.

General Extend Face (GEF)

Gegeben sei ein Graph $G = (V, E)$ mit Einbettung $\Phi : V \to \mathbb{R}^2$, eine Facette f und ein Polygon p. Gesucht ist eine Einbettung $\Psi : V \to \mathbb{R}^2$ mit den folgenden Eigenschaften:

1. p passt in $\Psi(f)$, es existiert also ein $u \in \mathbb{R}^2$ mit $\mathcal{I}(T_u(p)) \subset \mathcal{I}(\Psi(f))$.
2. Ψ minimiert die Gesamtkantenlängenänderung $\Delta(\Phi, \Psi)$ bezüglich Φ:

$$\Delta(\Phi, \Psi) = \sum_{e \in E} |\text{len}_\Phi(e) - \text{len}_\Psi(e)|$$

Satz 2 *GEF ist $\mathcal{NP}$-schwer.*

Beweis: Wir führen diesen Beweis analog zum vorherigen, indem wir das Problem Monotone3SAT auf das zugehörige Entscheidungsproblem GEF′ mit $\Delta(\Phi, \Psi) = 0$ reduzieren. Eine Instanz $\varphi = (X, C = C^+ \cup C^-)$ von Monotone3SAT ist dabei von der gleichen Form wie die PlanarMonotone3SAT-Instanzen. Wir verzichten lediglich auf die Forderung eines planaren Variable-Klausel-Graphen. Da folglich Monotone3SAT eine Verallgemeinerung von PlanarMonotone3SAT ist, folgt aus dieser Reduktion die $\mathcal{NP}$-Schwere von GEF.

Wir konstruieren nun einen Variable-Klausel-Graphen wie beispielhaft links in Abbildung 3.11 dargestellt ist. Dabei seien die Variablen oben nebeneinander und die Klauseln aus C^+ rechts (C^- links) untereinander angeordnet. Des Weiteren sei jede Klausel mit all ihren zugehörigen Variablen durch orthogonale Kanten verbunden. Nun legen wir für jede Variable links alle ausgehenden vertikalen Kanten zusammen, die zu C^--Klauseln führen. Analog ergibt sich rechts ein Kanal für alle Kanten zu C^+-Klauseln (Abbildung 3.11 in der Mitte). Ausgehend von diesem Layout konstruieren wir nun die GEF′-Instanz ψ indem wir wieder alle Elemente durch geeignete Gadgets ersetzen (Abbildung 3.11 rechts).

Zugrunde liegt ein genügend großer rechteckiger Bereich mit einer entsprechend großen rechteckigen Aussparung, der in Abbildung 3.11 rechts grau dargestellt ist. Dieser Bereich sei wieder starr in dem Sinne, dass es nur eine einzige zulässige längenerhaltende Einbettung gibt. Die grauen Knoten im grauen Bereich seien mit

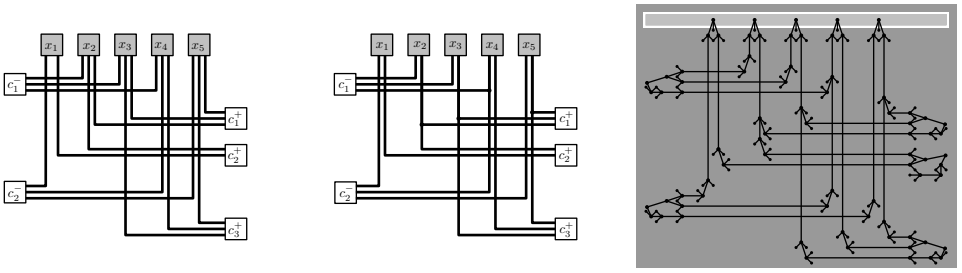

Abbildung 3.11: Der Variable-Klausel-Graph einer Instanz von MONOTONE3SAT (links), der gleiche Graph mit zusammengelegten Kanten (Mitte) und die daraus konstruierte GEF′-Instanz (rechts).

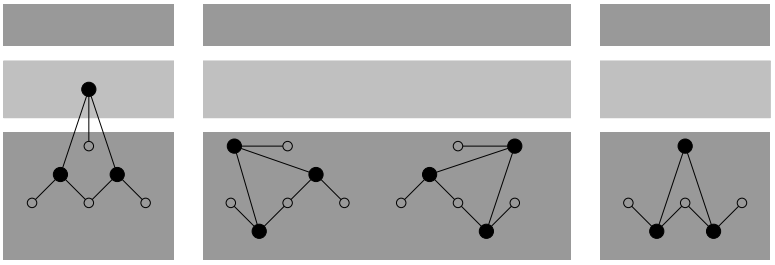

Abbildung 3.12: Das Variablen-Gadget in Ausgangsstellung (links), in seinen beiden zulässigen Konfigurationen (Mitte) und einer unzulässigen Anordnung (rechts).

dem grauen Bereich starr verbunden und können somit als unbeweglich angesehen werden. Dies alles kann erreicht werden, wenn die Ecken des grauen Bereichs, die grauen Knoten und eventuell zusätzliche Steiner-Knoten dreifach zusammenhängend verbunden werden. In der Aussparung des grauen Bereiches befindet sich das freizumachende Polygon, das hellgrau dargestellt ist. Alle schwarz dargestellten Knoten sind beweglich und nur durch die dargstellten Kanten mit anderen Knoten verbunden.

Entlang des freizumachenden Rechtecks ordnen wir nun nebeneinander für jede Variable ein Variablen-Gadget an. Dieses Gadget besteht aus drei nebeneinander liegenden starren Knoten, die durch zwei bewegliche Brückenknoten miteinander verbunden sind. An diesen beiden hängt ein weiterer Brückenknoten, der in den freizumachenden Bereich hineinragt und mit einem weiteren starren Knoten verbunden ist (Abbildung 3.12 links).

Jeder der beiden unteren Brückenknoten kann prinzipiell nach unten geklappt werden. Wird nur einer der beiden nach unten geklappt, so ergibt sich für den dritten beweglichen Knoten eine zulässige Lage (Abbildung 3.12 in der Mitte). Werden beide nach unten geklappt, so fallen der obere bewegliche und der obere starre Knoten zusammen und verletzen somit die Längenbedingung an dieser Kante. Diese Konfiguration ist also nicht zulässig (Abbildung 3.12 rechts). Wir halten fest:

A1: Ein Klausel-Gadget kann genau dann zulässig eingebettet werden, wenn genau ein Signal zur Variable hin zeigt.

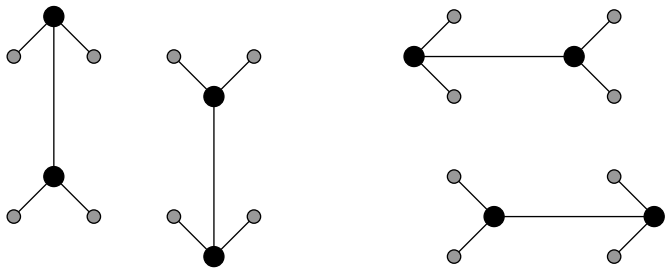

Abbildung 3.13: Das vertikale (links) und horizontale (rechts) Kanten-Gadget in seinen zulässigen Konfigurationen.

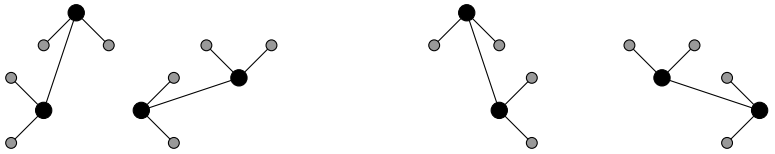

Abbildung 3.14: Das Turn-Gadget für negative (links) und positive Signale (rechts).

Wir interpretieren die Position dieser beiden Brückenknoten wieder als Signale im folgenden Sinne: Ist der rechte (linke) untere Brückenknoten nach oben geklappt, so zeigt das positive (negative) Signal zur Variable hin. Ist er nach unten geklappt, so zeigt das entsprechende Signal von der Variable weg.

Um diese Signale entlang der Kanten weiterzuleiten, ersetzen wir alle geradlinigen Kantensegmente durch Kanten-Gadgets. Diese Gadgets bestehen aus zwei Paaren starrer Knoten die durch jeweils einen beweglichen Brückenknoten verbunden sind. Diese Brückenknoten sind durch eine weitere Kante gekoppelt. Die vertikale und horizontale Variante ist in jeweils beiden zulässigen Einbettungen in Abbildung 3.13 dargestellt.

Die horizontalen und vertikalen Kanten-Gadgets, die Kantensegmente einer einzigen Kante repräsentieren, müssen nun noch miteinander verbunden und gekoppelt werden. Dafür verwenden wir das Lenk-Gadget, das aus einem horizontalen und einem vertikalen Signaltripel besteht. Die Brückenknoten werden wieder durch eine weitere Kante gekoppelt. Da die rechts angeordneten C^+-Klauseln nur aus positiven Literalen bestehen, müssen die rechts angeordneten positiven Signale nur nach rechts abgelenkt werden. Analog müssen negative Signale nur nach links zu den C^--Klauseln geleitet werden. In Abbildung 3.14 sind beide Versionen in ihren jeweiligen zulässigen Konfigurationen dargestellt. Wir halten fest:

A2: Für Kanten- und Lenk-Gadgets zeigt das jeweils ausgehende Signal genau dann von der Variable weg, wenn auch es das eingehende tut.

Abschließend betrachten wir nun das Klauel-Gadget. Es besteht aus vier untereinander angeordneten starren Knoten, die durch drei Brückenknoten verbunden sind. Diese Brückenknoten sind die eingehenden Signale. Das untere Signal wird nach hinten verschoben und dort nach oben umgelenkt, Die oberen beiden Signale sind durch

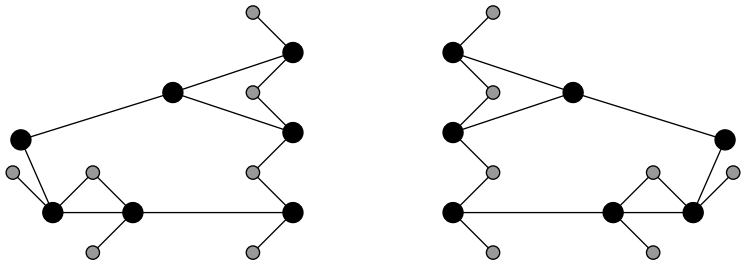

Abbildung 3.15: Das Klausel-Gadget für negative Klauseln (links) und positive Klauseln (rechts).

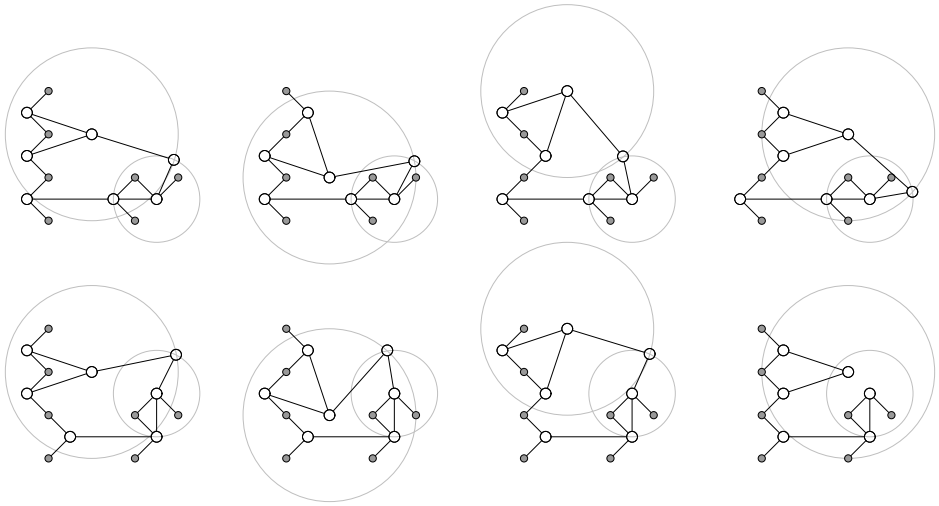

Abbildung 3.16: Die acht möglichen Kombinationen der Eingangssignale. Nur die letzte (unten rechts) hat keine zulässige Einbettung.

einen weiteren Brückenknoten miteinander verbunden, der mit dem umgelenkten unteren Signal über einen weiteren Brückenknoten verbunden ist (Abbildung 3.15).

Jedes der eingehenden Signale kann entweder zur Variable hin oder von der Variable weg zeigen. Die resultierenden acht Kombinationen sind in Abbildung 3.16 dargestellt. Zeigen alle Signale von den Variablen weg (unten rechts), so gibt es für den Brückenknoten keine zulässige Position. In allen anderen Fällen lässt sich das Gadget zulässig einbetten. Wir halten also fest:

A3: Das Klausel-Gadget ist genau dann zulässig, wenn wenigstens ein eingehendes Signal zur Variable hin zeigt.

Für den Sonderfall in der eine Klausel nur von zwei der Variablen abhängt, können wir im Klausel-Gadget wieder ein eingehendes Signal fixieren um zu erzwingen, dass mindestens eines der beiden anderen zur Variable hin zeigt. In Abbildung 3.17 ist das Klausel-Gadget mit fixiertem dritten Eingang dargestellt.

Für die Korrektheit der Reduktion bleibt wieder zu zeigen: Die konstruierte GEF′-Instanz ψ ist genau dann lösbar, wenn die zugrundeliegende Monotone3SAT-Formel φ erfüllbar ist.

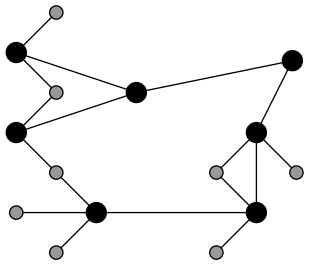

Abbildung 3.17: Das Klausel-Gadget mit einem fixierten Eingang.

Sei also zunächst ψ lösbar und Ψ eine lösende Einbettung. Laut Aussage A1 zeigt für jedes Variablen-Gadget genau ein Signal zur Variable hin. Wir belegen die Variable x_i genau dann als wahr, wenn für das zugehörige Variablen-Gadget das positive Signal zur Variable hin zeigt. Wir zeigen nun, dass diese Variablenbelegung die MONOTONE3SAT-Formel φ erfüllt.

Sei $c \in C$ eine beliebige Klausel. Nach Aussage A3 zeigt wenigstens ein eingehendes Signal des zugehörigen Klausel-Gadgets zur Variable hin. Nach Aussage A2 wird dieses Signal bis zum entsprechenden Variablen-Gadget propagiert. Ist $c \in C^+$, so ist dies nach Konstruktion das positive Signal dieses Variablen-Gadgets und nach gewählter Variablenbelegung ist diese Variable wahr, die Klausel c also in diesem Fall erfüllt. Für $c \in C^-$ ist dies analog das negative Signal und die Variable ist entsprechend negativ belegt, die Klausel c also auch in diesem Fall erfüllt. Insgesamt sind damit alle Klauseln erfüllt und die gefundene Variablenbelegung erfüllt φ.

Sei nun umgekehrt die MONOTONE3SAT-Formel φ erfüllbar und eine erfüllende Variablenbelegung fixiert. Für jede positive (negative) Variable klappen wir nun das negative (positive) Signal vom zugehörigen Variablen-Gadget von der Variable weg. Für jede positive (negative) Variable zeigt also das positive (negative) Signal zur Variablen hin. Nach Aussage A2 setzen sich diese Signale bis zu den Klausel-Gadgets fort. Sei nun $c \in C$ eine beliebige Klausel. Da unsere Variablenbelegung erfüllend ist, muss mindestens ein beteiligtes Literal wahr sein. Ist $c \in C^+$, so ist die entsprechende Variable wahr und nach Konstruktion zeigt das positive Signal von diesem Variablen-Gadget zur Variable hin. Für $c \in C^-$ ist analog die Variable negativ belegt und das negative Signal zeigt zur Variable hin. In beiden Fällen zeigt also wenigstens ein eingehendes Signal des Klausel-Gadgets zur Variable hin und nach Aussage A3 folgt, dass alle Klausel-Gadgets zulässig eingebettet werden könnnen. Insgesamt ist somit die konstruierte Einbettung also eine Lösung der MONOTONE3SAT-Instanz ψ. ■

3.3 Monotones Expandieren

Zum Abschluss dieses Abschnittes betrachten wir noch eine Variante von GEF, die sich durch die zusätzliche Einschränkung ergibt, dass die orthogonalen Knotenordnungen erhalten bleiben müssen. Anschaulich bedeutet dies, dass die horizontale und vertikale Ordnung aller Knoten im Ergebnisgraph der des Ausgangsgraphen entsprechen muss. Insbesondere bleiben also achsenparallele Kanten achsenparallel. Diese zusätzliche Einschränkung soll dabei helfen die mental map zu erhalten [ELMS95]

und könnte etwa für UML-Diagramme besonders geeignet sein, da dort häufige orthogonale Kantenzüge verwendet werden.

Monotone Extend Face (MEF)

Gegeben sei ein planarer Graph $G = (V, E)$ mit planarer Einbettung $\Phi : V \to \mathbb{R}^2$, eine Facette f und ein Polygon p. Gesucht ist eine Einbettung $\Psi : V \to \mathbb{R}^2$ mit den folgenden Eigenschaften:

1. Φ und Ψ haben gleiche orthogonale Ordnungen, für $v, w \in V$ gilt also:

$$\begin{aligned} \Phi(v)_x < \Phi(w)_x \quad &\Leftrightarrow \quad \Psi(v)_x < \Psi(w)_x \\ \Phi(v)_y < \Phi(w)_y \quad &\Leftrightarrow \quad \Psi(v)_y < \Psi(w)_y \end{aligned}$$

2. p "passt" in $\Psi(f)$, es existiert also ein $u \in \mathbb{R}^2$ mit $\mathcal{I}(T_u(p)) \subset \mathcal{I}(\Psi(f))$.
3. Ψ minimiert die Gesamtkantenlängenänderung $\Delta(\Phi, \Psi)$ bezüglich Φ:

$$\Delta(\Phi, \Psi) = \sum_{e \in E} |\text{len}_\Phi(e) - \text{len}_\Psi(e)|$$

Satz 3 *MEF ist $\mathcal{NP}$-schwer.*

Beweis: Wie in den zwei vorherigen Beweisen lässt sich auch diese $\mathcal{NP}$-Schwere durch analoge Reduktion von PlanarMonotone3SATzeigen. Um die einzelnen Gadgets unabhängig bewegen zu können, ordnet man diese auf einer Diagonalen an und verbindet die entsprechenden Signale durch achsenparallele Kanten. Die Konstruktion und Argumentation ist den bisherigen sehr ähnlich und bringt keine neuen Erkenntnisse. Wir werden den Beweis hier deshalb nicht weiter ausführen. ■

4. Heuristiken

Im Folgenden wollen wir uns nun mit zwei heuristische Algorithmen befassen. Zum einen betrachten wir eine nichtuniforme Skalierung, die zwar eine zulässige Lösung garantiert, den Graphen jedoch stark verzerren kann. Zum anderen wollen wir uns einen Spring-Embedder anschauen, der zwar nicht auf allen Instanzen eine korrekte Lösung liefern kann, auf dünnen Graphen jedoch qualitativ gute Ergebnisse erzielt. Um die Argumentationen zu vereinfachen, werden wir im Folgenden das Polygon und die Facette als konvex annehmen. Beide Verfahren können aber auch für den nicht konvexen Fall angepasst werden.

4.1 Nichtuniforme Skalierung

Die grobe Idee für das folgende Verfahren besteht darin, mehrere Skalierungsrichtungen auszuwählen und den Graph in diesen Richtungen unterschiedlich stark zu skalieren. Dafür wird ein Skalierungszentrum c verwendet, das sich sowohl im Inneren des freizumachenden Polygons p, als auch im Inneren der umschließenden Facette f befinden sollte. Dieses Zentrum sollte sich dabei möglichst mittig in der Facette befinden. Der Schwerpunkt sollte für die meisten Instanzen gute Ergebnisse liefern. Zu jeder Skalierungsrichtung α_i gehört ein Skalierungsfaktor λ_i. Für jeden Graph-Knoten x_i werden dann die benachbarten Skalierungsrichtungen α_i und α_j ausgewälht und die Skalierungsfaktoren λ_i und λ_j linear im Winkel interpoliert.

Da sich auf diese Weise die Knoten der Facette f vom Skalierungszentrum c wegschieben lassen, können wir eine zulässige Lösung erhalten, indem wir durch jeden Facettenknoten f_i eine Skalierungsrichtung α_i legen und die Skalierungsfaktoren genügend groß wählen. Bei dieser Wahl der Skalierungsfaktoren müssen wir im Wesentlichen zwei Arten von Problemen lösen um zu einer zulässigen Konfiguration zu gelangen:

- Knotenproblem: Ein Facettenknoten liegt vor einer Polygonkante und somit innerhalb des Polygons (Abbildung 4.1 links).
- Kantenproblem: Ein Polygonknoten liegt hinter einer Facettenkante und damit ein Teil der Kante im Inneren des Polygons (Abbildung 4.2 links).

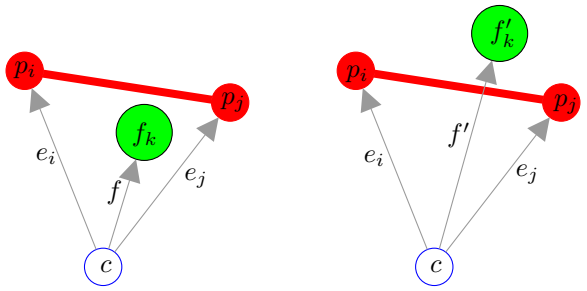

Abbildung 4.1: Ein Knotenproblem zwischen Facettenknoten f_k und Polygonkante (p_i, p_j) (links) und eine mögliche Lösung (rechts).

Bei dieser Sprechweise liegt ein Knoten hinter einer Kante, wenn er vom Zentrum aus gesehen von dieser Kante verdeckt wird. Er liegt vor einer Kante, wenn er sich in dem von dieser Kante und dem Zentrum aufgespannten Dreieck befindet.

Sind alle Knoten- und Kantenprobleme behoben, so liegen alle Graphknoten und Graphkanten komplett außerhalb des Polygons. Das Ergebnis ist also eine zulässige Einbettung.

Um ein Knotenproblem zu identifizieren, seien konkret (p_i, p_j) eine Polygonkante und f_k ein Facettenknoten. Wir betrachten die Vectoren $e_i = p_i - c$, $e_j = p_j - c$ und $f = f_k - c$. Da c im Inneren des konvexen Polygons liegt, sind e_i und e_j linear unabhängig und wir können f als Linearkombination $f = ue_i + ve_j$ darstellen. Ein Knotenproblem kann nur dann vorliegen, wenn f_k im Inneren des Winkels $\angle p_i c p_j$ liegt, also $u \geq 0$ und $v \geq 0$ gilt. In diesem Fall liegt genau dann ein Knotenproblem vor, wenn f_k im Inneren des Dreiecks $\triangle p_i c p_j$ liegt, also wenn $u + v < 1$ gilt. Um dieses Knotenproblem zu beheben, müssen wir dann die λ_k geeignet wählen. Nach der Skalierung liegt f_k an der Position $f'_k = c + \lambda_k f$, also

$$f' = f'_k - c = \lambda_k f = \lambda_k u e_i + \lambda_k v e_j = u' e_i + v' e_j$$

mit $u' = \lambda_k u$ und $v' = \lambda_k v$. Um das Knotenproblem zu lösen, muss f'_k außerhalb des Dreiecks $\triangle p_i c p_j$ liegen, also $1 \leq u' + v' = \lambda_k (u + v)$ gelten. Folglich müssen wir $\lambda_k \geq 1/(u + v)$ wählen.

Um ein Kantenproblem zu identifizieren, sei (f_i, f_j) eine Facettenkante und p_k ein Polygonknoten. Nun betrachten wir die Vektoren $d_i = f_i - c$, $d_j = f_j - c$ und $p = p_k - c$. Wir stellen wieder p als Linearkombination von d_i und d_j dar: $p = ud_i + vd_j$. Ein Kantenproblem kann nur vorliegen, wenn p im Winkel $\angle f_i c f_j$ liegt, also wenn $u \geq 0$ und $v \geq 0$ gilt. In diesem Fall liegt genau dann ein Kantenproblem vor, wenn p_k außerhalb des Dreiecks $\triangle f_i c f_j$ liegt, also wenn $u + v > 1$ gilt. Um dieses Kantenproblem zu beheben, müssen wir λ_i und λ_j geeignet wählen. Nach der Skalierung liegen f_i und f_j an den Positionen $f'_i = c + \lambda_i d_i$ und $f'_j = c + \lambda_j d_j$, also sind $d'_i = \lambda_i d_i$ und $d'_j = \lambda_j d_j$. Wir stellen p wieder als Linearkombination dar:

$$p = ud_i + vd_j = u\frac{d'_i}{\lambda_i} + v\frac{d'_j}{\lambda_j} = u' d'_i + v' d'_j$$

mit $u' = u/\lambda_i$ und $v' = v/\lambda_j$. Um das Kantenproblem zu lösen, muss p_k innerhalb des Dreiecks $\triangle f'_i c f'_j$ liegen, also $1 \geq u' + v' = u/\lambda_i + v/\lambda_j$ gelten. Ist also einer der

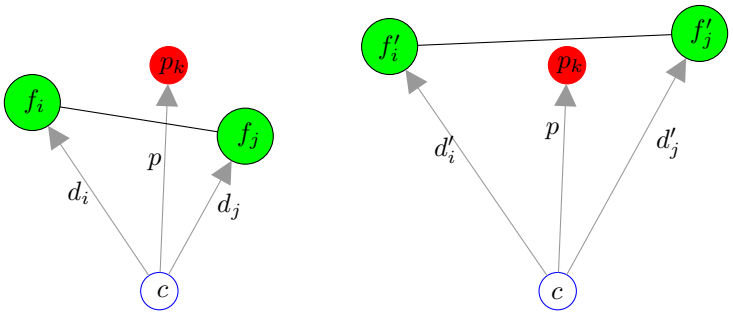

Abbildung 4.2: Ein Kantenproblem zwischen Facettenkante (f_i, f_j) und Polygonknoten p_k (links) und eine mögliche Lösung (rechts).

beiden Faktoren gegeben, ergibt sich für den anderen entsprechend $\lambda_i \geq u\lambda_j/(\lambda_j - v)$ beziehungsweise $\lambda_j \geq v\lambda_i/(\lambda_i - u)$.

Unsere Aufgabe besteht nun darin, geeignete Werte für die λ_i zu finden, die diese Ungleichungen erfüllen und dabei eine gewisse Zielfunktion minimieren. Wir werden dieses quadratische Programm mit quadratischen Nebenbedingungen mit einem iterativen Verfahren approximieren. Ausgangspunkt ist eine beliebige zulässige Belegung der λ_i. Eine solche erhält man etwa, indem man jedes Knotenproblem mit $\lambda_i \geq 1/(u+v)$ und jedes Kantenproblem mit $\lambda_i \geq u+v$ und $\lambda_j \geq u+v$ löst und für jedes λ_i das Maximum dieser Schranken wählt.

Unser Verfahren iteriert nun zirkulär um die Facette und versucht den Wert der Zielfunktion durch lokale Änderungen zu verbessern. An jedem Facettenknoten f_i wird dabei der zugehörige Skalierungsfaktor λ_i verändert und die benachbarten Skalierungsfaktoren gegebenenfalls vergrößert um zu einer neuen zulässigen Belegung zu gelangen. Hat diese neue Konfiguration einen besseren Wert für die Zielfunktion, so arbeiten wir mit dieser weiter, ansonsten verwerfen wir die Änderungen und bleiben bei der alten Belegung.

Für die von uns untersuchten Optimierungsfunktionen konvergiert diese Iteration sehr schnell gegen ein lokales Minimum. Nach etwa zwanzig Umläufen um unsere Facette ist das Minimum auf unseren Beispielen ausreichend stabil. Dabei haben wir die folgenden Zielfunktionen untersucht:

MF: Minimiere den maximalen Skalierungsfaktor.

MD: Minimiere den maximalen Abstand der Facettenknoten zum Zentrum.

SF: Minimiere die Summe aller Skalierungsfaktoren.

SD: Minimiere die Summe alle Abstände der Facettenknoten zum Zentrum.

A: Minimiere die Fläche der Facette.

U: Minimiere den Umfang der Facette.

Man beachte, dass die Minimierungen der Maxima MF und MD sogar in Polynomialzeit exakt lösen lassen. Betrachtet man nämlich jedes Paar benachbarter Skalierungsfaktoren $(\lambda_i, \lambda_{i+1})$ getrennt, so lässt sich eine lokale Lösung mit $\lambda_i = \lambda_{i+1}$

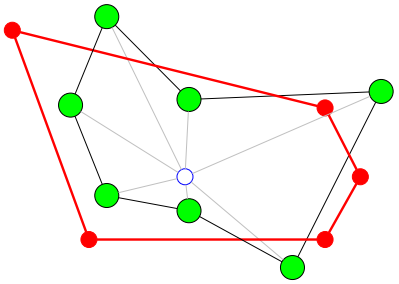

Abbildung 4.3: Eine Beispielinstanz: Die Facette muss vergrößert werden um das Polygon zu umschließen. Das Zentrum und die Skalierungsrichtungen sind dargestellt.

Solution	MF	MD	SF	SD	A	U
MF	2.4241	360.39	11.698	1615.8	251375	1604.4
MD	2.4241	292.14	11.538	1562.3	242886	1499.3
SF	2.4241	334.86	11.409	1531.7	239028	1502.5
SD	2.9445	353.73	11.669	1517.6	235757	1482.3
A	2.9548	333.60	11.609	1519.6	235266	1470.7
U	3.2876	317.58	12.446	1582.0	257450	1425.9

Tabelle 4.1: Werte der Zielfunktionen bei unterschiedlicher Optimierung der Beispielinstanz: In Zeile i sind die Werte aller Zielfunktionen für die Lösung dargestellt, die bei Optimierung der Zielfunktion i gefunden wurde.

finden die obige Ungleichungen erfüllt, also in dem aufgespannten Sektor zulässig ist. Diese λ_i liefern im Allgemeinen zwar keine globale Lösung, sind aber untere Schranken für den global maximalen Skalierungsfaktor. Das Maximum λ' all dieser unteren Schranken ist nun der gesuchte kleinstmögliche Wert des maximalen Skalierungsfaktors: Setzt man $\lambda_i = \lambda'$ für jede Skalierungsrichtung α_i, sind alle Knoten- und Kantenprobleme in allen Sektoren gelöst, die Lösung alse zulässig. Da λ' eine untere Schranke war, ist diese Lösung minimal. Die Zielfunktion MD minimiert man analog indem man benachtbarte Faktoren so wählt, dass die zugehörigen Facettenknoten gleichweit vom Zentrum entfernt sind. Dies liefert untere Schranken für den global maximalen Abstand. Dieser Abstand kann dann durch entsprechende Wahl der Faktoren auch erreicht werden. Wie sich herausstellen wird, sind diese Zielfunktionen den anderen jedoch qualitativ unterlegen.

In Abbildung 4.3 ist eine Beispielinstanz mit fünf Polygon- und sieben Facettenknoten dargestellt. Abbildung 4.4 zeigt die Ergebnisse unseres Verfahrens für jede Zielfunktion. In Tabelle 4.1 sind die Werte aller Zielfunktionen für alle Lösungen dargestellt. Die erste Zeile entspricht dabei der Lösung, die den maximalen Skalierungsfaktor minimiert, die zweite Zeile minimiert den maximalen Abstand. Entlang der Hauptdiagonalen befinden sich folglich die Spaltenminima.

Je nach Anwendung kann die eine oder andere Zielfunktion bevorzugt werden. Die SF und SL sind den MF und ML meist überlegen, da sie nicht nur an dem einen Maximum-Knoten optimieren, sondern an jedem Facettenknoten. Flächenminimie-

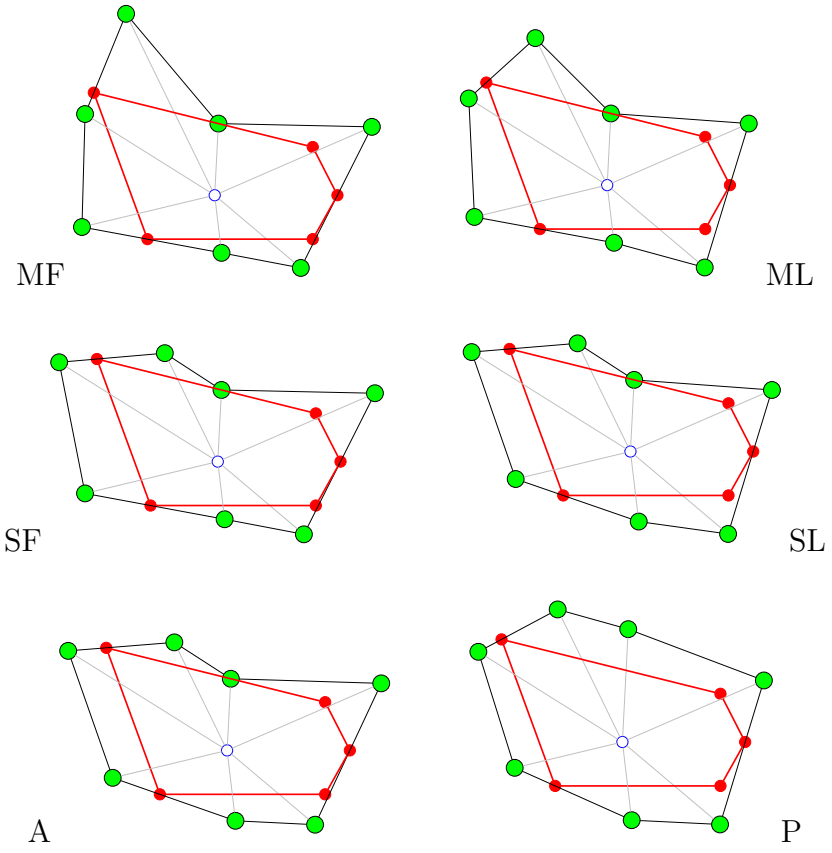

Abbildung 4.4: Die Ergebnisse unseres Verfahrens für die verschiedenen Zielfunktionen nach zehn Facettenumläufen.

rung kann erwünscht sein, um die Größe der resultierenden Zeichnung klein zu halten. Umfangminimierung hat die interessante Eigenschaft, dass die Facette immer konvex ist. Da wir diese Skalierungsrichtungen und Faktoren nun verwenden wollen um nicht nur die Problemfacette, sondern den kompletten Graph zu transformieren, kann es von Interesse sein diese Transformation möglichst homogen zu gestalten. Dies kann man etwa erreichen, indem man die Zielfunktion um Terme erweitert, die benachbarte Skalierungsfaktoren bestraft, wenn diese sehr unterschiedlich sind. Desweiteren könnten Stauchungen unerwünscht sein, was leicht durch weitere untere Schranken für die Skalierungsfaktoren behoben werden kann. Auf diese Erweiterungen werden wir nicht weiter eingehen.

Um einen kompletten Graphen mit unseren gefundenen Skalierungsfaktoren zu transformieren, suchen wir uns für jeden Knoten die benachbarten Skalierungsrichtungen in dessen Sektor sich der Knoten befindet, und interpolieren die Faktoren entlang des Winkels. In Abbildung 4.5 ist oben links eine Beispielinstanz dargestellt. Darunter ist der transformierte Graph zu sehen, nachdem er mit SF-Skalierungsfaktoren verzerrt wurde. Man beachte, dass die Skalierungsfaktoren in diesem Fall Werte von ungefähr zwei annehmen. Folglich wird der gesamte Graph in der Größe etwa verdoppelt.

Da wir aber eigentlich nur die problematische Facette vergrößern und den Rest des Graphen möglichst wenig verändern wollen, werden wir versuchen die weiter vom Zentrum entfernten Bereiche weniger stark zu verformen. Dazu werden wir entlang der Skalierungsrichtungen nicht mehr skalieren, sondern eher verschieben. Der zuge-

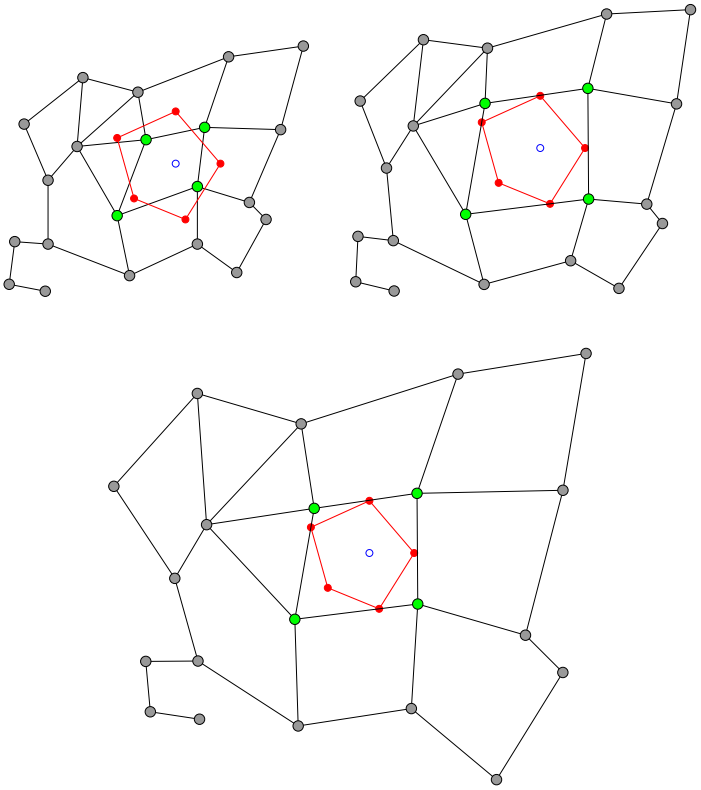

Abbildung 4.5: Eine Beispielinstanz (oben links), das Ergebnis der nichtuniformen SF-Skalierung (unten) und das Ergebnis der eingeschränkten SF-Skalierung (oben rechts). (Kürzel dieser Instanz: INW)

hörige Facettenknoten wird dabei so weit verschoben, wie er bisher skaliert wurde. Alle dahinterliegenden Knoten werden aber nicht mehr multiplikativ skaliert, sondern stattdessen additiv um diese Differenz vom Zentrum weggeschoben. Für zwischen den Skalierungsrichtungen liegende Knoten wird dieses Verhalten wie gehabt interpoliert. Dadurch erreichen wir, dass der komplette Graph sich im Durchmesser nur etwa so viel vergrößert wie die Problemfacette. Das Ergebnis dieser eingeschränkten Skalierung auf unserer Beispielinstanz ist in Abbildung 4.5 oben rechts zu sehen.

In Abbildung 4.6 ist links ein quadratischen Gitter dargestellt. In eine der Facetten soll nun ein recht großes Polygon eingefügt werden. Im Ergebnis der eingeschränkten A-Skalierung rechts kann man gut die gleichmäßige Interpolation der Verzerrung erkennen. Weiter entfernte Facetten werden dabei deutlich weniger stark verzerrt.

In Abbildung 4.7 ist nun der Einfluss der Facettenwahl dargestellt. In den gleichen Graph soll dabei das gleiche Polygon eingefügt werden (oben). Die unterschiedliche Wahl der zu vergrößernden Facette führt dann zu sehr verschiedenen Ergebnissen (unten). Im linken Fall wird in eine recht große fünfeckige Facette eingefügt. Das macht wenig Probleme und das Ergebnis ist kaum verzerrt. Im rechten Fall hingegen ist das Polygon in einem kleinen Dreieck unterzubringen, was nur mit deutlich sichtbaren Verzerrungen zu erreichen ist. Insgesamt ist aber auch diese Lösung durchaus akzeptabel.

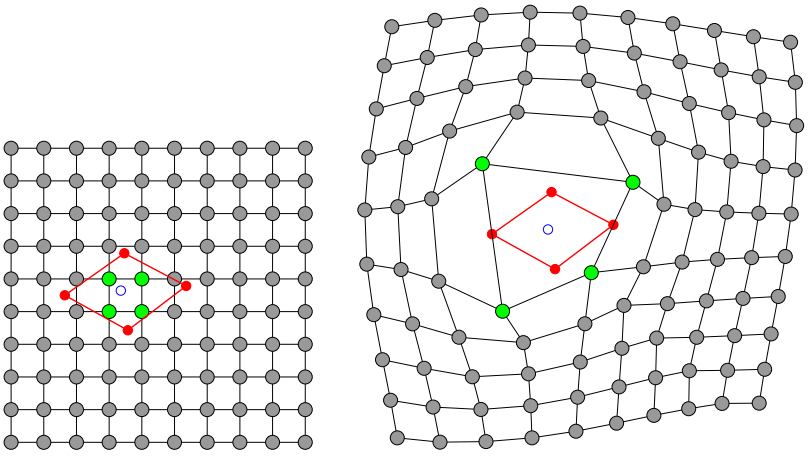

Abbildung 4.6: Ein quadratisches Gitter (links) wird gleichmäßig verzerrt um dem Polygon Platz zu schaffen (eingeschränkte A-Skalierung, rechts). (GRID)

4.2 Spring-Embedder

Bei diesem Verfahren betrachten wir die Knoten als Punktmassen in der Ebene, auf die gewisse Kräfte wirken. Eine physikalische Simulation beschleunigt und bewegt diese Punktmassen dann entsprechend der wirkenden Kräfte, um das System einem Kräftegleichgewicht anzunähern. Das Ergebnis dieser Simulation liefert dann ein Layout des ursprünglichen Graphen. Mit geeignet gewählten Kräften können auf dünnen Graphen auf diese Weise recht ansprechende Einbettungen erzeugt werden.

In klassischen Spring-Embeddern wird jede Kante zwischen zwei Knoten als eine Feder modelliert, die an den zugehörigen Punktmassen angebracht ist. Die daraus resultierenden Kräfte können die Knoten dann abhängig von der Federkonstanten verschieden stark zusammen ziehen, oder auseinander drücken. Mit geeigneten Federlängen und Federkonstanten lässt sich dadurch beispielsweise erreichen, dass alle Kanten versuchen, dieselbe Länge anzunehmen. Eine weitere oft verwendete Kraft wirkt abstoßend auf zu nah aneinander liegende Knoten, um einen gewissen Mindestabstand zu gewährleisten. Dies kann beispielsweise erwünscht sein, wenn eine gegebene Zeichenauflösung nicht unterschritten werden soll.

Um unser konkretes Problem zu approximieren, werden wir auch etwas speziellere Kräfte untersuchen: Wir betrachten zum einen *Polygonkräfte*, die sicherstellen sollen, dass die resultierende Einbettung zulässig ist. Zum anderen sollen *Stabilisierungskräfte* dafür sorgen, dass der Graph sich in einem gewissen Sinne nicht allzu sehr verformt, um die Wiedererkennbarkeit des ursprünglichen Graphen zu erhöhen. Dabei wird auch die Minimierung der Gesamtkantenlängenänderung eine Rolle spielen.

Um mit den Polygonkräften eine zulässige Einbettung zu erreichen, wollen wir wieder die bereits charakterisierten Knoten- und Kantenprobleme lösen. Diese Kräfte sollen die Problemknoten vom Zentrum aus über das Polygon hinaus schieben.

Liegt etwa ein Knotenproblem für die Polygonkante (p_i, p_j) und den Facettenknoten f_k vor, so betrachten wir die Vektoren $p = p_j - p_i$ und $f = f_k - c$ (Abbildung 4.8

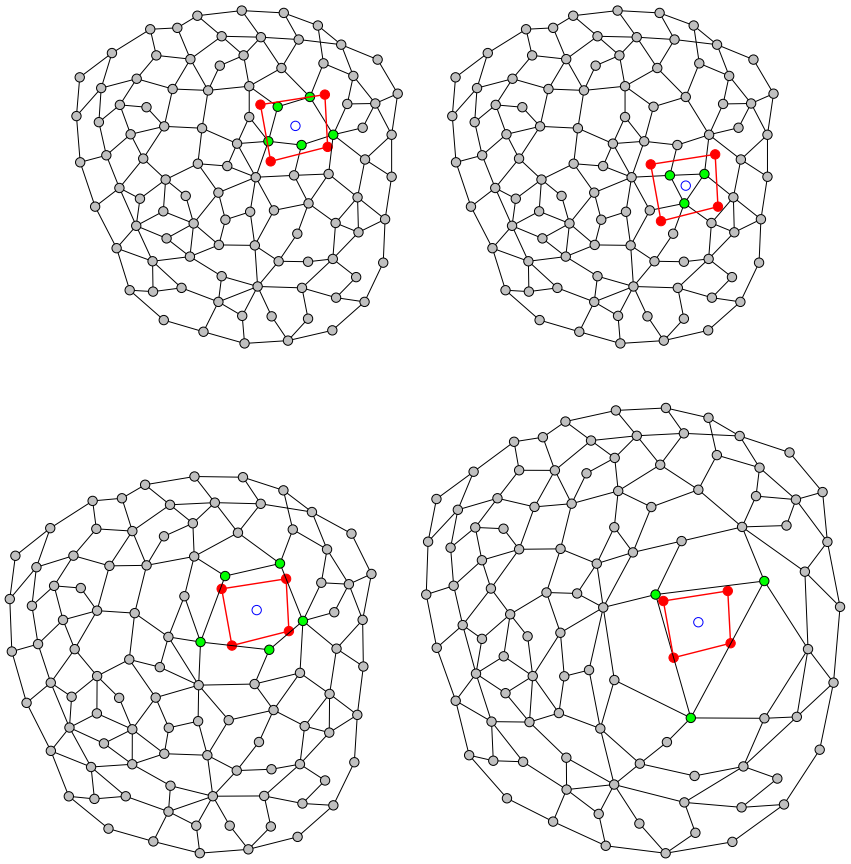

Abbildung 4.7: Der Einfluss der Zielfacette. (links BPF, rechts BPD)

links). Sei $x = c + \beta f$ der Schnittpunkt der Kante mit dem Zentrumsstrahl mit $\beta \geq 1$. Dann ist x gerade genau der Punkt, an den wir den Knoten f_k schieben wollen. Die Kraft wirkt also in Richtung f und ihr Betrag sollte proportional zur Auslenkung $|x - f_k| = (\beta - 1)|f|$ sein. Wir können dies als Feder zwischen f_k und x modellieren, die Länge 0 anstrebt.

Für ein Kantenproblem zwischen Facettenkante (f_i, f_j) und Polygonknoten p_k betrachten wir die Vektoren $f = f_j - f_i$ und $p = p_k - c$ (Abbildung 4.8 rechts). Um das Problem zu lösen, wollen wir die Facettenkante genügend weit in Richtung p verschieben. Ist $x = c + \beta p$ der Schnittpunkt der Kante mit dem Zentrumstrahl mit $\beta \leq 1$, so ist $p_k - x = (1 - \beta)p$ gerade der Vektor um den wir f_i und f_j verschieben wollen. Die wirkenden Kräfte zeigen also in Richtung p mit Betrag proportional zu $(1 - \beta)|p|$. Wie auch beim Knotenproblem modellieren wir dies durch zwei Federn der Länge 0.

Lässt man diese Polygonkräfte nun auf die Facettenknoten einer Instanz wirken, so wird eine genügend lange Simulation unseres Spring-Embedders tatsächlich die Knoten- und Kantenprobleme lösen, da alle problematischen Facettenknoten so lange vom Zentrum weg bewegt werden, bis dies der Fall ist. In Abbildung 4.9 ist eine Beispielinstanz und das Ergebnis dieser Simulation dargestellt.

Dies liefert bereits eine zulässige Einbettung, deren Qualität im Allgemeinen aber noch nicht zufriedenstellend ist. Dies liegt im Wesentlichen daran, dass lediglich die problematischen Facettenknoten aus dem Polygon herausgeschoben werden, wäh-

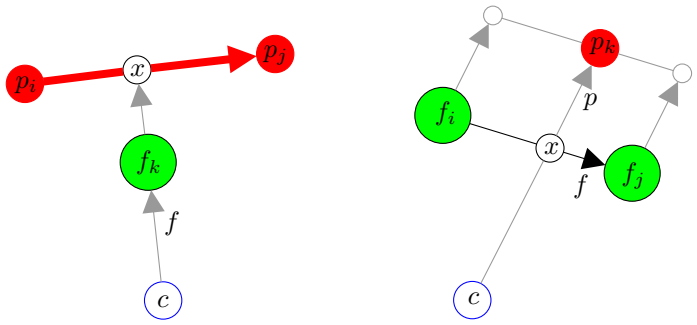

Abbildung 4.8: Ein Knotenproblem (links) und ein Kantenproblem (rechts) mit den jeweils beteiligten Polygonkräften.

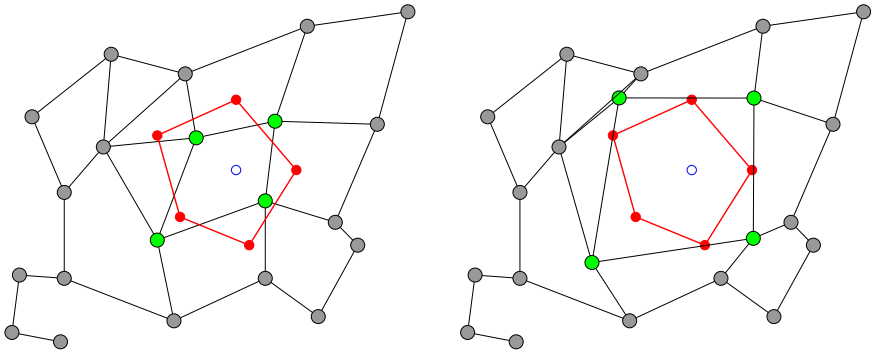

Abbildung 4.9: Eine Beispielinstanz (links) und das Ergebnis des Spring-Embedders mit Polygonkräften (rechts).

rend der Rest des Graphen unverändert bleibt. Dadurch kommt es insbesondere in den benachbarten Facetten zu starken Verformungen und Stauchungen. Um diesen Effekten entgegen zu wirken und die Wiedererkennbarkeit des Graphen zu verbessern, führen wir Stabilisierungskräfte ein.

Um die Gesamtkantenlängenänderung zu minimieren und dabei auch den Stauchungen entgegenzuwirken, erzeugen wir für jede Graphkante eine Feder zwischen den inzidenten Knoten, die die Länge der Kante im ursprünglichen Layout anstrebt. Diese *Kantenkräfte* schieben also gestauchte Kanten auseinander und ziehen gestreckte Kanten zusammen. Um die Facette dabei nicht ins Polygon zurück zu ziehen, sollten die beteiligten Federkräfte um Größenordnungen kleiner sein, als die der Polygonkräfte.

Der Einfluss dieser Kräfte auf unserer Beispielinstanz ist links in Abbildung 4.10 zu sehen. Man erkennt deutlich, dass die Kanten anliegender Facetten versuchen ihre ursprünglichen Längen anzunehmen, während das Polygon nach wie vor freigemacht wird. Desweiteren illustriert dieses Ergebnis ein weiteres Problem: Eine Facette oben links vom freizumachenden Polygon ist umgeklappt. Mit den bisherigen Kräften ist dieses Phänomen aber wenig überraschend, da eine gute Approximation für das Kräftegleichgewicht sich diese Umklappungen zu nutze machen kann. Insbesondere lassen sich auf diese Art nämlich sehr kompakte Layouts finden, die die Längenanfor-

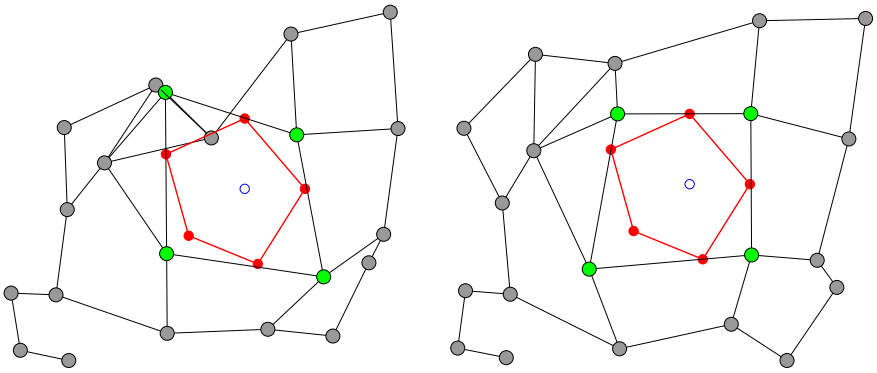

Abbildung 4.10: Ergebnis des Spring-Embedders mit zusätzlichen Kantenkräften (links) und Winkelkräften (rechts).

derungen gut erfüllen. Da wir der Wiedererkennbarkeit wegen die kombinatorische Einbettung aber möglichst beibehalten wollen, werden wir nun eine weitere Kraft einführen, die diesen Umklappungen entgegen wirken soll.

Im Beispielgraph liegt der Extremfall vor, dass ein Paar benachbarter Graphkanten seine Orientierung ändert: Die eine Kante hat während der Simulation die andere überstrichen. Betrachten wir nun also ein beliebiges Paar $p_{ijk} = (e_{ij}, e_{ik}) = ((v_i, v_j), (v_i, v_k))$ adjazenter Kanten und den dazwischen liegendenden, mathematisch positiv orientierten Winkel $\angle_{ijk} = \angle e_{ij} e_{ik}$, so wollen wir möglichst sicherstellen, dass dieser Winkel seinen positiven Wertebereich $(0, \pi)$ beziehungsweise negativen Wertebereich $(-\pi, 0)$ nicht verlässt.

Dafür benutzen wir eine *Winkelkraft*, die wie ein Drehmoment auf die Kanten wirkt, um den ursprünglichen Winkel wiederherzustellen. Ist beispielsweise der Winkel zu klein geworden, so werden die äußeren Knoten v_j und v_k jeweils senkrecht zu ihrer Kante e_{ij} beziehungsweise e_{ik} auseinandergedrückt. Die jeweils entgegengesetzte Kraft wirkt pro Kante auf den gemeinsamen Knoten v_i. In Summe schieben diese Kräfte diesen Knoten dann in den Winkel hinein. Der Betrag der zu den Kanten senkrechten Einzelkräfte sollte dabei proportional zur Winkeldifferenz sein. Diese Kräfte für einen Winkel der vergrößert werden soll, sind links in Abbildung 4.11 dargestellt. Ist anderenfalls der Winkel zu groß geworden, so werden genau die zum vorherigen Fall entgegengesetzten Kräfte dafür sorgen, dass der Winkel wieder vergrößert wird. Die Kräfte an Knoten v_j und v_k zeigen nun senkrecht nach innen, und die beiden Kräfte an v_i zeigen nach außen. In Summe wird v_i dadurch aus dem Winkel herausgeschoben.

Der Einfluss der Winkelkräfte auf das Simulationsergebnis unserer Beispielinstanz ist rechts in Abbildung 4.10 zu sehen. Wie erhofft konnte das Umklappen der zuvor problematischen Facette verhindert werden. Dabei ist das Polygon frei und die Kanten im Kontraktionsbereich haben ähnliche Längen wie im Ausgangsgraphen. Mit den bisher beschriebenen Kräften lassen sich bereits Einbettungen finden, die das Problem lösen und dabei dem ursprünglichen Graphlayout sehr ähnlich sind.

Bevor wir diese Ergebnisse evaluieren, wollen wir uns noch eine weitere Variation ansehen. Sehr große Graphen müssen mitunter nur sehr lokal verformt werden, um das Polygon freizumachen. Mit unseren bisherigen Kräften ist es aber sehr wahr-

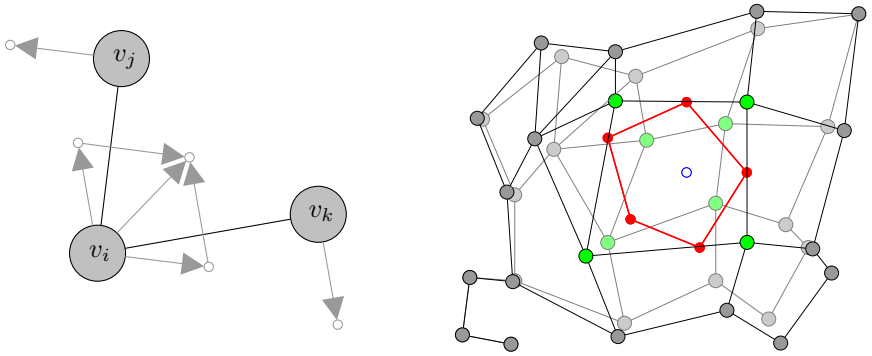

Abbildung 4.11: Die auf ein zusammengedrücktes Kantenpaar wirkenden Winkelkräfte. Das Ergebnis einer Simulation mit Knotenkräften (links). Vor dem hinterlegten ursprünglichen Layout ist erkennbar, dass sich die am weitesten vom Zentrum entfernten Knoten nicht bewegt haben (rechts).

scheinlich, dass sich der komplette Graph während der Simulation etwas dreht oder verschiebt. Da dies je nach Anwendung unerwünscht sein kann, wollen wir auch diesem Effekt entgegen wirken.

Dazu führen wir eine weiter Kraft ein, die jeden Knoten an seine alte Position zurückziehen soll. Diese *Knotenkräfte* werden durch eine Feder zwischen aktueller und ursprünglicher Knotenposition modelliert, die Länge 0 anstrebt. Da wir mit dieser Kraft die Simulation in der Nähe des freizumachenden Polygons nicht stören, aber weiter entfernte Knoten an ihren Positionen halten wollen, werden wir die zugehörigen Federkonstanten mit zunehmender Entfernung vom Zentrum exponentiell steigern. Auf unserer Beispielinstanz lässt sich der Effekt bereits am Randknoten oben rechts und an den drei Pfadknoten unten links erkennen. Im Gegensatz zu vorher sind diese Knoten nicht von ihrer ursprünglichen Position abgewichen (Abbildung 4.11 rechts).

Ein typisches Problem kräftebasierter Verfahren ist die Wahl der einzelnen Federkonstanten. Wir haben bereits gesehen, dass die Polygonkräfte auf jeden Fall dominieren sollten, um sicherzustellen dass das Polygon auch freigemacht wird. Die mit der Entfernung exponentiell zunehmenden Knotenkräfte verhalten sich auch vernünftig: Im Zentrum können sie vernachlässigt werden und außerhalb des Problembereiches halten sie den Graphen fest. Es bleibt die Frage, wie Kanten- und Winkelkräfte zu wählen sind und welchen Einfluss sie jeweils haben. Dazu betrachten wir exemplarisch die Ergebnisse des Spring-Embedders für verschiedene Parameterkombinationen am Beispiel der Gitter-Instanz aus Abbildung 4.6: In Abbildung 4.12 sind sechs Lösungen dargestellt.

Im ersten Bild links oben dominieren die Kantenkräfte und die Winkelkräfte sind vernachlässigbar klein. Folglich versuchen alle Kanten ihre alte Länge beizubehalten und es kommt zu willkürlich anmutenden Verschiebungen. Im zweiten Bild bekommen die Winkelkräfte einen sehr kleinen Einfluss, der bereits ausreicht die Gitterzeilen und -spalten etwas geradezubiegen. In den Bereichen der Facettenknoten gibt es aber noch sehr starke Stauchungen, die im dritten Bild durch den immer prominenteren Einfluss der Winkelkräfte erstmals aufgelöst werden können. Im vierten

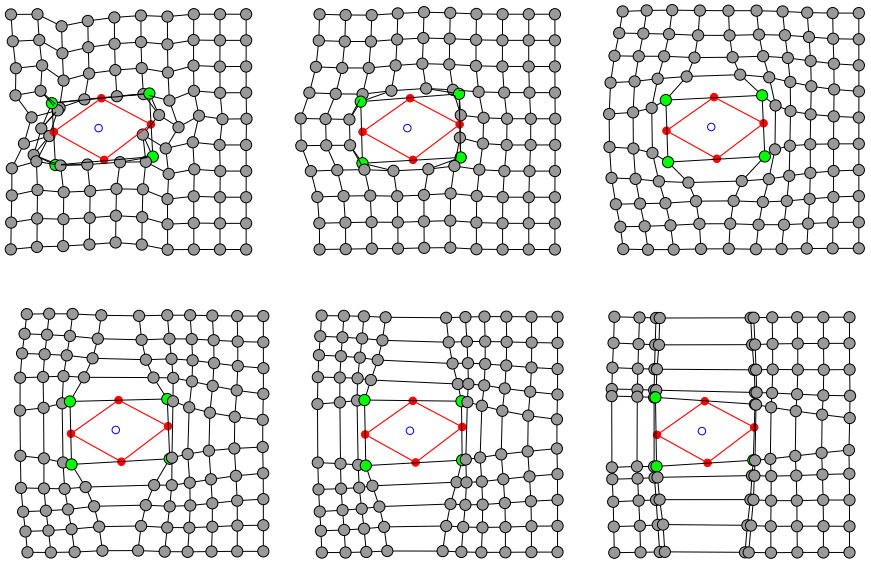

Abbildung 4.12: Der Einfluss der Kanten- und Winkelkräfte. Links dominieren die Kantenkräfte, rechts die Winkelkräfte. Die zugehörigen Δ^{rel}-Werte: 0.077, 0.078, 0.16, 0.24, 0.32, 0.41.

und fünften Bild richten die Winkelkräfte die Gitterkanten immer mehr an den alten Richtungen aus, bis sie im sechsten Bild rechts unten fast achsenparallel sind. In diesem Bild dominieren die Winkelkräfte und die Kantenkräfte sind vernachlässigbar klein. Auch hier kommt es zu starken Stauchungen in den Bereichen der Facettenknoten. Die Knotendichte ist hier am inhomogensten. Qualitativ sind die Lösungen drei und vier den anderen überlegen, da dort einerseits die Gitterstruktur gut erhalten bleibt und andererseits auch die Knotenverteilung recht homogen ist. In den anderen Bildern kollidieren die Facettenknoten mit benachbarten Graphknoten. Quantitativ betrachten wir unsere ursprüngliche Minimierungsfunktion der Gesamtkantenlängenänderung Δ. Um auch verschiedene Graphen vergleichen zu können, betrachten wir die relative Änderung Δ^{rel}, die sich aus der absoluten Änderung Δ ergibt, wenn man durch die Gesamtkantenlänge im Ausgangsgraphen teilt. Für die dargestellten Lösungen ergeben sich die in der Bildunterschrift aufgelisteten Werte. Links oben verändert sich die Gesamtkantenlänge also um knapp acht Prozent, rechts unten um etwa 41. Es ist wenig überraschend, dass diese Werte mit dem Einfluss der Kantenkräfte korrelieren, da die Kantenkräfte gerade diese Änderung minimieren soll. Wir sehen hier aber auch, dass minimale Δ-Werte im Allgemeinen nicht den qualitativ besten Lösungen entsprechen.

In Abbildung 4.7 hatten wir den Einfluss der Zielfacette für die nichtuniforme Skalierung betrachtet. In Abbildung 4.13 sind Lösungen des Spring-Embedders für beide Instanzen dargestellt. Man kann gut erkennen, dass die Knotenkräfte die linke Kontur des Graphen komplett starr halten konnten. Allerdings sind diese Knotenkräfte auch verantwortlich dafür, dass die Dreiecksfacette rechts nicht groß genug werden kann, um das Polygon aufzunehmen. Man sieht hier deutlich, dass der Spring-Embedder keine Lösungsgarantien geben kann. Je nach Parameterwahl kann dieses Problem unterschiedlich stark ausgeprägt sein.

Dieses kräftebasierte Verfahren lässt sich sehr einfach für nichtplanare Graphen ver-

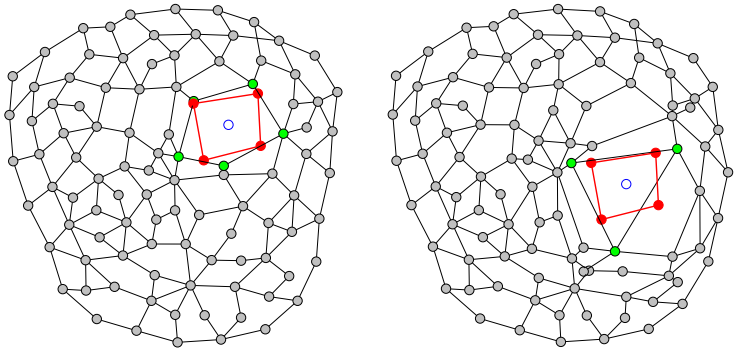

Abbildung 4.13: Der Einfluss der Zielfacette auf den Spring-Embedder.

allgemeinern. Man muss lediglich bei der Erkennung und Behebung der Knoten- und Kantenprobleme anstatt der Facettenknoten und Facettenkanten alle Graphknoten und Graphkanten betrachten. In Abbildung 4.14 ist ein nicht planar eingebetteter Graph (oben) mit den entsprechenden Ergebnissen (unten) dargestellt. Die verschiedenen Polygone links und rechts veranschaulichen dabei den Einfluss des Polygons auf das Ergebnis. Auch hier lässt sich in beiden Fallen gut der alte Graph wiedererkennen.

4.3 Vergleich

Nachdem wir unsere beiden heuristischen Verfahren nun schon einzeln evaluiert haben, wollen wir sie an dieser Stelle etwas allgemeiner vergleichen. Dazu betrachten wir noch einmal die jeweiligen Ergebnisse der nichtuniformen Skalierung (NS) und des Spring-Embedders (SE) auf den betrachteten Instanzen INW, GRID, BPF und BPD. Da wir die nichtuniforme Skalierung nicht auf nichtplanare Problemfacetten verallgemeinert haben, liegen für die Instanzen NPLS und NPLB nur Ergebnisse des Spring-Embedders vor.

In Tabelle 2 sind quantitave Ergebnisse dargestellt. Pro Instanz und Verfahren sind die relativen Gesamtkantenlängenänderungen Δ^{rel} und die Laufzeiten T in Millisekunden dargestellt. Die Längenänderungen sind dabei relativ zur Gesamtkantenlänge im Ausgangsgraphen. Die Zeiten wurden auf einem 2.4 GHz Core i5 gemessen. An den Zahlen lassen sind im Wesentlichen zwei Dinge ablesen: Der Spring-Embedder liefert bezüglich der Längenänderung bessere Lösungen, ist aber um mehrere Größenordnungen langsamer. Genauere Betrachtungen suggerieren, dass der Spring-Embedder bezüglich Δ umso überlegener ist, je größer der zugrundeliegende Graph ist. So kommen beide Verfahren auf etwa zwanzig Prozent Kantenlängenänderung im kleineren Graphen INW, während der Spring-Embedder für die größeren Graphen BPF und BPD etwa um Faktor zwei besser ist. Es ist auch wenig überraschend, dass das kräftebasierte Verfahren hier überlegen ist, da es einen deutlich größeren Lösungsraum durchsucht. Während die nichtuniforme Skalierung nur einen Freiheitsgrad pro Facettenknoten hat, gibt es hier zwei Freiheitsgrade pro Graphknoten. In Abbildung 4.6 sieht man beispielsweise deutlich, dass die Facette das Polygon nur in zwei Ecken berührt. Eine bessere Lösung ergibt sich, wenn die Facette das Polygon in allen vier Ecken berührt, wie etwa in Abbildung 4.12. Solch eine Lösung kann aber

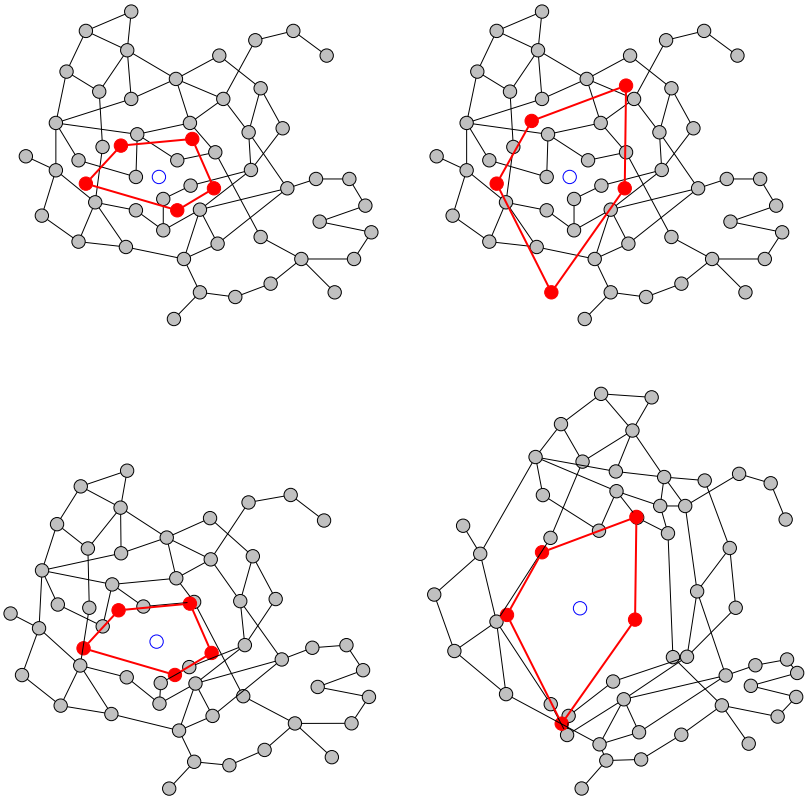

Abbildung 4.14: Der Einfluss des Polygons auf einem nicht planar eingebetteten Graphen. (links NPLS, rechts NPLB)

mit dem Skalierungsansatz nicht gefunden werden, da alle Facettenknoten auf ihren Zentrumsstrahl eingeschränkt sind. Diesen Vorteil bezahlt der Spring-Embedder aber mit einer deutlich höheren Laufzeit. Die Suche der Skalierungsfaktoren ist linear in der Anzahl der Facettenknoten (zur Evaluation wurde jede Facette zwanzig Mal umlaufen). Die anschließende Transformation des Graphen ist linear in der Anzahl der Graphknoten. Die Laufzeit des Spring-Embedders hingegen ist quadratisch in der Anzahl der Graphknoten (zur Evaluation wurden 1024 Iterationen simuliert). Diese Korrelation zwischen Knotenanzahl und Laufzeit ist auch deutlich in den gemessenen Zeiten zu erkennen.

Qualitativ lassen sich mit beiden Verfahren vergleichbar gute Ergebnisse erzielen. Die Wahl passender Parameter für den Spring-Embedder ist dabei jedoch nicht zu vernachlässigen. Für die nichtuniforme Skalierung hingegen ist nur eine geeignete Zielfunktion zu wählen. Die vorgeschlagenen Kandidaten liefern dabei auf den untersuchten Instanzen gute Lösungen. Die Gewichtungen der Federkräfte bei dem Spring-Embedder hingegen hat einen sehr großen Einfluss auf die Güte der Ergebnisse, wie wir in Abbildung 4.12 gesehen hatten. Wird etwa die Polygonkraft nicht dominant genug gewählt, liefert die Simulation im Allgemeinen nicht einmal eine zulässige Lösung, wie etwa in Abbildung 4.13.

Zusammenfassend ist die nichtuniforme Skalierung unserem Spring-Embedder für allgemeine Instanzen deulich überlegen. Um mehrere Größenordnungen schneller werden robust gute und vor allem zulässige Lösungen gefunden. Unter größerem Konfigurations- und Zeitaufwand lassen sich mit dem Spring-Embedder typischer-

Instanz	$\Delta^{\text{rel}}_{\text{NS}}$	$\Delta^{\text{rel}}_{\text{SE}}$	T_{NS}	T_{SE}
INW	0.21	0.20	0.05	336
GRID	0.39	0.24	0.06	1032
BPF	0.14	0.08	0.09	868
BPD	0.35	0.17	0.04	952
NPLS	-	0.08	-	226
NPLB	-	0.31	-	262

Tabelle 4.2: Vergleich der nichtuniformen Skalierung (NS) und des Spring-Embedders (SE). Dargestellt sind pro Instanz und Verfahren die relative Gesamtkantenlängenänderung (Δ^{rel}) und die Laufzeit (T) in ms.

weise etwas bessere Ergebnisse erzielen, allerdings ohne Lösungsgarantien. Es darf dabei aber auch nicht vergessen werden, dass sich der Spring-Embedder deutlich feiner steuern lässt. So ist dieses Verfahren viel leichter durch neue Kräfte an zusätzliche Anforderungen anpassbar. Beispielsweise konnten wir weiter entfernte Knoten an ihren Positionen halten, indem wir die Knotenkräfte eingeführt haben.

5. Zusammenfassung

Zum Abschluss dieser Arbeit wollen wir die wichtigsten Ergebnisse hier kurz zusammentragen. Im Komplexitätsabschnitt haben wir gesehen, dass das von uns betrachtete Problem des Platzschaffens in seinen drei untersuchten Varianten $\mathcal{NP}$-schwer ist. Für die planare und nichtplanare Version haben wir dies mit einer Reduktion von PlanarMonotone3SATbeziehungsweise Monotone3SATgezeigt. Anschließend haben wir zwei Heuristiken konstruiert, die bei der Approximation helfen sollen. Bei der nichtuniformen Skalierung wird dabei der Graph von einem Zentrum weg in verschiedene Richtungen unterschiedlich stark skaliert, beziehungsweise verschoben. Dieses Verfahren liefert schnell und robust garantiert zulässige Lösungen. Das zweite Verfahren basiert auf einem Spring-Embedder mit speziellen Kräften, die zum einen eine zulässige Lösung herbeiführen, zum anderen aber auch gewisse ästhetische Kriterien optimieren sollen. Dieser Ansatz liefert etwas bessere Ergebnisse, ist der nichtuniformen Skalierung aber generell nicht vorzuziehen, da der Spring-Embedder deutlich sensiblere Konfiguration erfordert und um mehrere Größenordnungen langsamer ist.

Für spezielle Szenarien kann der Spring-Embedder recht einfach durch zusätzliche Kräfte an neue Anforderungen angepasst werden. Mit seinen vielen Freiheitsgraden für die einzelnen Kräfteverhältnisse lässt er sich auch sehr genau steuern. Beim Skalierungsverfahren hingegen könnten Kombinationen der bisherigen und weiterer Zielfunktionen interessante Ergebnisse liefern. Da wir dieses Verfahren nur für planare Graphen beschrieben haben, wäre es auch interessant, es für den nichtplanaren Fall zu verallgemeinern. Eine weitere mögliche Verbesserung der Ergebnisse ließe sich durch Kombination beider Heuristiken erreichen. So könnte man etwa zunächst mit der nichtuniformen Skalierung schnell eine zulässige und bereits gute Lösung finden, die dann noch durch einige Iterationen des Spring-Embedders optimiert werden könnte. Somit ließe sich der Tradeoff zwischen Qualität und Laufzeit je nach Anforderung einstellen.

Literaturverzeichnis

[BW97] U. Brandes und D. Wagner: *A bayesian paradigm for dynamic graph layout.* In: *Graph Drawing*, Seiten 236–247. Springer, 1997.

[CDBT+92] R.F. Cohen, G. Di Battista, R. Tamassia, I.G. Tollis und P. Bertolazzi: *A framework for dynamic graph drawing.* In: *Proceedings of the eighth annual symposium on Computational geometry*, Seiten 261–270. ACM, 1992.

[dBK08] M. de Berg und A. Khosravi: *Finding perfect auto-partitions is NP-hard.* In: *Proc. 25th European Workshop Comput. Geom.(EuroCG'08)*, Seiten 255–258, 2008.

[DG02] S. Diehl und C. Görg: *Graphs, they are changing.* In: *Graph Drawing*, Seiten 23–31. Springer, 2002.

[DGK01] S. Diehl, C. Görg und A. Kerren: *Preserving the mental map using foresighted layout.* In: *Proceedings of Joint Eurographics–IEEE TCVG Symposium on Visualization (VisSym'01)*, Seiten 175–184, 2001.

[DH96] R. Davidson und D. Harel: *Drawing graphs nicely using simulated annealing.* ACM Transactions on Graphics (TOG), 15(4):301–331, 1996.

[EKP04] C. Erten, S. Kobourov und C. Pitta: *Intersection-free morphing of planar graphs.* In: *Graph Drawing*, Seiten 320–331. Springer, 2004.

[ELMS95] P. Eades, W. Lai, K. Misue und K. Sugiyama: *Layout adjustment and the mental map.* Journal of Visual Languages and Computing, 6(2):183–210, 1995.

[FE02] C. Friedrich und P. Eades: *Graph drawing in motion.* J. Graph Algorithms Appl., 6(3):353–370, 2002.

[FR06] M. Freire und P. Rodríguez: *Preserving the mental map in interactive graph interfaces.* In: *Proceedings of the working conference on Advanced visual interfaces*, Seiten 270–273. ACM, 2006.

[KR92] D.E. Knuth und A. Raghunathan: *The problem of compatible representatives.* Arxiv preprint cs/9301116, 1992.

[LLY06] Y.Y. Lee, C.C. Lin und H.C. Yen: *Mental map preserving graph drawing using simulated annealing.* In: *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation-Volume 60*, Seiten 179–188. Australian Computer Society, Inc., 2006.